Obstacle Avoiding Robot

2023

Author : Team 5

1. Project Introduction 3
1.1. Car Components 3
1.2. System Requirements 3
1.3. Assumptions 4
2. High Level Design 5
2.1. System Architecture 5
2.1.1. Definition 5
2.1.1.1. MCAL 5
2.1.1.1. APP Layer 5
2.1.2. Layered Architecture 5
2.2.1. Digital Input Output Module 6
2.2.2. External Interrupt Module 6
2.2.3. Timer Module 7
2.2.4. Push-Button Module 7
2.2.5. PWM Module 7
2.2.6. Input Capture Unit Module 7
2.2.7. Ultrasonic Module 7
2.2.8. LCD Module 8
2.2.9. Keypad Module 8
2.2.10. DC Motor Module 8
2.2.11. SystemModules 9
2.3. APIs 9
2.3.1. Definition 9
2.3.2. MCAL_APIs 10
2.3.2.1. DIO 10
2.3.2.2. EXI 11
2.3.2.3. Timer 13
2.3.2.4. ICU 14
2.3.2.5. PWM 16
2.3.3. ECUAL_APIs 17
2.3.3.1. Push-BUTTON 17
2.3.3.2. Keypad 18
2.3.3.3. LCD 19
2.3.3.4. DCM 21
2.3.3.5 Ultrasonic 23
2.3.4. Application_APIs 24
2.3.4.1. Car_operations 24
2.3.4.2. App 25
3. Low Level Design 26
3.1. MCAL 26
3.1.1. DIO 26
3.1.1.1. DIO_init 26

3.1.1.2. DIO_writepinn..... 27
3.1.1.3. DIO_readpin..... 28
3.1.1.4. DIO_togglepin..... 28
3.1.2. EXI..... 28
3.1.2.1 EXT_vINTERRUPT_Init..... 29
3.1.2.2 EXT_vINTERRUPT_Denit..... 30
3.1.2.3 EXT_vINTERRUPT_setSenseControl..... 31
3.1.2.3.1 EXT_vINTERRUPT_setSenseControl_EXT0..... 33
3.1.2.3.2 EXT_vINTERRUPT_setSenseControl_EXT1..... 33
3.1.2.3.3 EXT_vINTERRUPT_setSenseControl_EXT2..... 34
3.1.2.4 EXT_INTERRUPT_SetInterruptHandler..... 36
3.1.2.5 EXT0_INTERRUPT_SetInterruptHandler..... 38
3.1.2.6 EXT1_INTERRUPT_SetInterruptHandler..... 40
3.1.2.7 EXT2_INTERRUPT_SetInterruptHandler..... 41
3.1.3. Timer..... 42
3.1.3.1. TIMER_TMR2NormalodeInit..... 43
3.1.3.2. TIMER_TMR2ISR..... 43
3.1.3.3 TIMER_vdStop..... 45
3.1.3.4. TIMER_TMR2Start..... 46
3.1.3.4. TIMER_TMR1NormaModeInit..... 47
3.1.3.4. Timer_TMR1Stop & ISR(TIM1_OVF_INT)..... 48
3.1.3.5 TIMER_TMR1Start..... 49
3.1.4. ICU..... 50
3.1.4.1 ICU_risingEgeCapture..... 50
3.1.4.2. ICU_getValue..... 52
3.1.4.3. ICU_enablePIE..... 53
3.1.4.4. ISR(EXT_INT_2)..... 54
3.1.4.1..... 54
3.1.5. PWM..... 55
3.1.5.1 TIMER0_init..... 55
3.1.5.2 TIMER0_start..... 55
3.1.5.3 TIMER0_stop..... 55
3.1.5.4 TIMER0_initPWM..... 56
3.1.5.5 TIMER0_setPwm..... 56
3.1.5.6 TIMER0_PWM_ExecutedFunction..... 57
3.2. ECUAL..... 59
3.3.1. Push-BUTTON..... 59
3.3.1.1 PUSH_BTN_intialize..... 59
3.3.1.2 PUSH_BTN_read_state..... 60
3.3.2. Keypad..... 61
3.3.2.1 KEYPAD_init..... 62
3.3.2.2 KEYPAD_getButton..... 62

3.3.3. LCD..... 64
3.3.3.1.LCD_Init..... 64
3.3.3.2.LCD_write_ins..... 65
3.3.3.3.LCD_write_data..... 65
3.3.3.4.LCD_write_string..... 66
3.3.3.5.LCD_write_number..... 66
3.3.3.6.LCD_write_Char..... 67
3.3.3.7.LCD_Clear..... 68
3.3.3.8.LCD_Setcursor..... 68
3.3.3.9.LCD_init_pins..... 69
3.3.4. DCM..... 70
3.3.4.1.DCM_motorInit..... 70
3.3.4.2.DCM_changeDirection..... 71
3.3.4.3.DCM_vdStop..... 71
3.3.4.4.DCM_setDutyCycle..... 72
3.3.4.5.DCM_rotate..... 73
3.3.4.6. DCM_MoveForward / DCM_MoveBackward..... 75
3.3.5 Ultrasonic..... 76
3.3.5.1 ultrasonic_vInit..... 76
3.3.5.2 ultrasonic_vGetDistance..... 77
3.3. App..... 78
3.3.1. Car_operations..... 79
3.3.1.1.SetDefaultRotation..... 79
3.3.1.2.ObstacleMoreThan70..... 80
3.3.1.3.ObstacleMoreThan30..... 80
3.3.1.4.ObstacleMoreThan20..... 81
3.3.1.5.ObstacleLessThan20..... 82
3.3.1.6.LCD_update..... 84
3.3.1.7.Car_stop..... 85
3.3.2. App..... 85
3.3.2.1.app_init..... 85
3.3.2.2.app_main..... 86
4. Precompiling and linking configuration..... 87
4.1.DIO..... 87
4.1.1.linking Configuration..... 87
4.1.2.Precompiling Configuration..... 88
4.2.LCD..... 90
4.2.1.Precompiling Configuration..... 90
4.3.Timer..... 91
4.3.1 linking Configuration..... 91
4.3.2 Precompiling Configuration..... 92
4.4. ICU..... 94

4.4.1 linking Configuration....94
4.4.2 Precompiling Configuration.... 95
4.5. DCM....96
4.5.1 Linking Configuration....96
4.5.1 Precompiling Configuration.... 96
4.6. PWM.... 97
4.6.1 Linking Configuration....97
4.6.2 Precompiling Configuration.... 97
4.7. Keypad....98
4.7.1 Linking Configuration....98
4.7.2 Precompiling Configuration.... 99
4.8 External Interrupt....100
4.8.1 Precompiling Configuration.... 100
4.8.2 Linking Configuration....102
4.9 Push Button.... 103
4.9.1 Precompiling Configuration.... 103
4.9.2 Linking Configuration....103
4.10 Ultrasonic....104
4.10.1 Linking Configuration....104

1. Project Introduction

1.1. Car Components

- ATmega32 Microcontroller.
- Four Motors (M1, M2, M3, M4).
- One Push Button To Change The Direction Of Rotation.
- Keypad.
- One Ultrasonic Sensor.
- LCD.

1.2. System Requirements

- The car **starts initially** from **0 speed.**
- The default rotation direction is to the **right**
- Press **PB2** to start or stop the robot
- **After Pressing Start:**
    a. **The LCD will display a centred message in line 1 “Set Def. Rot.”**
    b. **The LCD will display the selected option in line 2 “Right”**
    c. **The robot will wait for 5 seconds to choose between Right and Left**
    d. **When PB1 is pressed once, the default rotation will be Left and the LCD line 2 will be updated**
    e. **When PB1 is pressed again, the default rotation will be Right and theLCD line 2 will be updated**
    f. ***For each press the default rotation will changed and the LCD line 2 is updated***
    g. **After the 5 seconds the default value of rotation is set**
- The robot will move **after 2 seconds** from setting the default direction of rotation.
- F**or No obstacles or object is far than 70 centimetres:**
    a. **The robot will move forward with 30% speed for 5 seconds**
    b. **After 5 seconds it will move with 50% speed as long as there was no object or objects are located at more than 70 centimetres distance**
    c. **The LCD will display the speed and moving direction in line 1: “Speed:00% Dir: F/B/R/S”, F: forward, B: Backwards, R: Rotating, and S: Stopped**
    d. **The LCD will display Object distance in line 2 “Dist.: 000 Cm”**
    e. **For Obstacles located between 30 and 70 centimetres**
- **For Obstacles located between 30 and 70 centimetres**.
    a. The robot will decrease its speed to 30%
    b. LCD data is updated
- **For Obstacles located between 20 and 30 centimetres**
    a. The robot will stop and rotates 90 degrees to right/left according to the chosen configuration
    b. The LCD data is updated
- **For Obstacles located less than 20 centimetres**
    a. The robot will **stop**, move **backwards** with **30% speed** until distance is **greater than 20 and less than 30**
    b. The LCD data is updated
    c. **Then perform point 8**
- **Obstacles surrounding the robot (Bonus)**

a. **If the robot rotated for 360 degrees without finding any distance greater than 20 it will stop**
b. **LCD data will be updated.**
c. **The robot will frequently (each 3 seconds) check if any of the obstacles was removed or not and move in the direction of the furthest object**

1.3. Assumptions

2. High Level Design

2.1. System Architecture

- Layered architecture in embedded systems organises software components into distinct layers, each with a specific responsibility and level of abstraction.
- The layers are arranged hierarchically, starting from the hardware layer, followed by MCAL, HAL and ending with the application layer.
- Layering provides benefits like modularity, scalability, and maintainability by separating software components into distinct layers with clear separation of concerns.

2.1.1. Definition

2.1.1.1. MCAL

- MCAL is a layer in embedded systems that provides a standardised interface to the microcontroller hardware.
- It includes functions/drivers for controlling hardware peripherals and is critical for enabling higher-level software layers to be developed independently of the hardware platform.

2.1.1.1. ECUAL

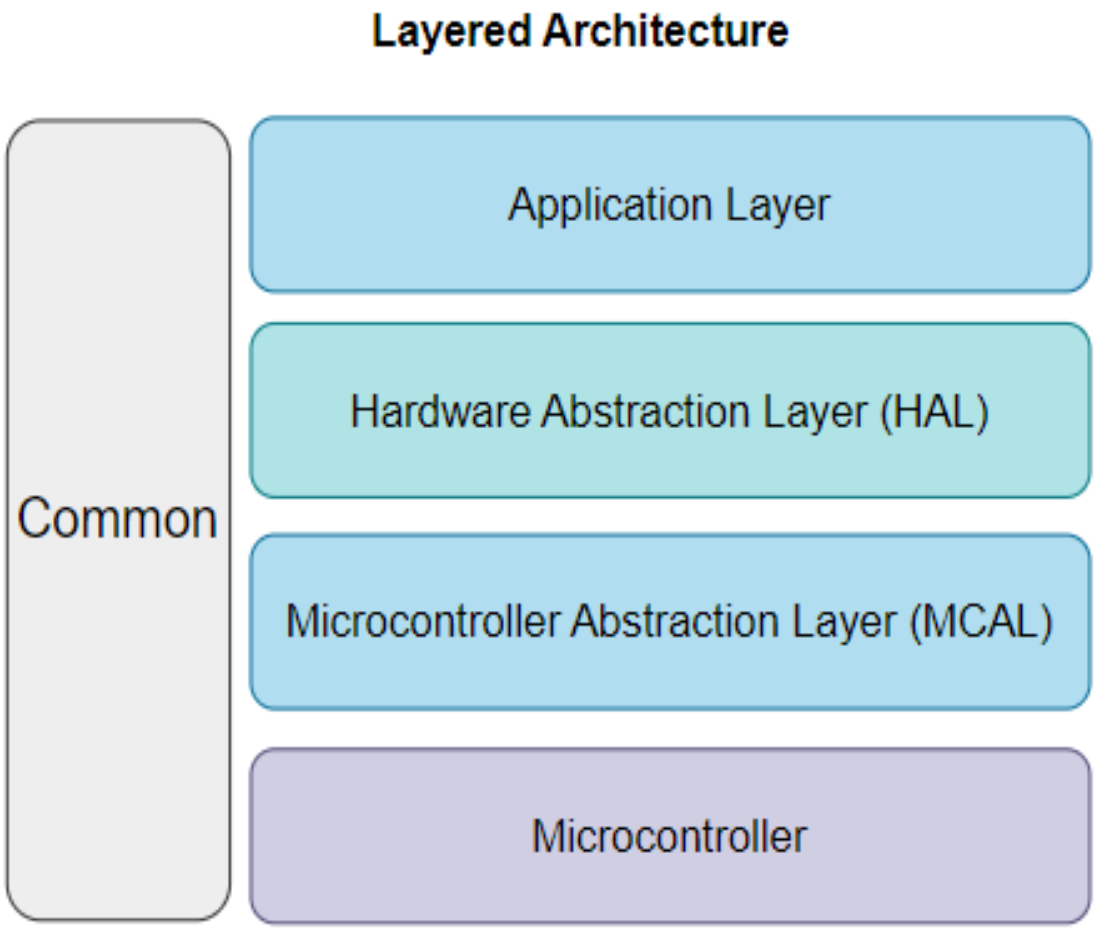
- In a layered architecture for embedded systems, the ECU (Electronic Control Unit) layer sits above the MCAL layer and provides a higher-level, more application-specific interface for controlling the system's functions.
- It implements the system's primary functionality and interacts with the MCAL layer through well-defined APIs and interfaces. The ECU layer may also include functionality for managing system diagnostics, error handling, and communication with other ECUs or external systems

2.1.1.1. APP Layer

- In a layered architecture for embedded systems, the APP (Application) layer is responsible for providing the system's primary functionality and

interacts with the lower layers through well-defined APIs and interfaces. It may include software components for implementing specific functions, managing system diagnostics, error handling, and communication with other systems.

2.1.2. Layered Architecture



2.2. Modules Descriptions

2.2.1. Digital Input Output Module.

- This module deals with the digital input and output operations, such as reading and writing to digital pins of a microcontroller or a microprocessor. It may include functions for setting pin direction, reading and writing digital values, and handling interrupts related to digital pins.

2.2.2. External Interrupt Module.

- Push buttons are designed to be easy to use and can come in many shapes, sizes, and colours, making them suitable for a wide range of applications. They work by completing an electrical circuit when pressed, and are commonly used in electronics, machinery, and appliances.

2.2.3. Timer Module.

- The *TIMER* module is responsible for generating timing events that are used by other modules in the system. It provides a set of APIs to configure the timer clock source and prescaler, set the timer mode (count up/down), set the timer period, enable/disable timer interrupts, and define an ISR that will be executed when the timer event occurs .

2.2.4. Push-Button Module.

- Is a feature in microcontrollers that allows an external signal to interrupt the normal program execution and perform a specific task. It is a hardware feature that is triggered by a change in the state of an external signal, such as a button press or a sensor trigger.

2.2.5. PWM Module.

- The pulse width modulation (PWM) module enables the generation of pulse width modulated signals on GPIO. The module implements an up or up-and-down counter with four PWM channels that drive assigned GPIOs .

2.2.6. Input Capture Unit Module.

- Input Capture unit that can capture external events and give them a time-stamp indicating time of occurrence. The external signal indicating an event, or multiple events, can be applied via the ICP1 pin or alternatively, via the analog-comparator unit.

2.2.7. Ultrasonic Module.

- This module handles the ultrasonic sensor and communicates with the application layer to provide distance readings of objects in front of the car. It also triggers the car to stop or change direction based on the distance readings.

2.2.8. LCD Module.

- This display is used to show the current status of the car and distance readings from the ultrasonic sensor. It is controlled by the application layer and communicates with the microcontroller using supporting drivers .

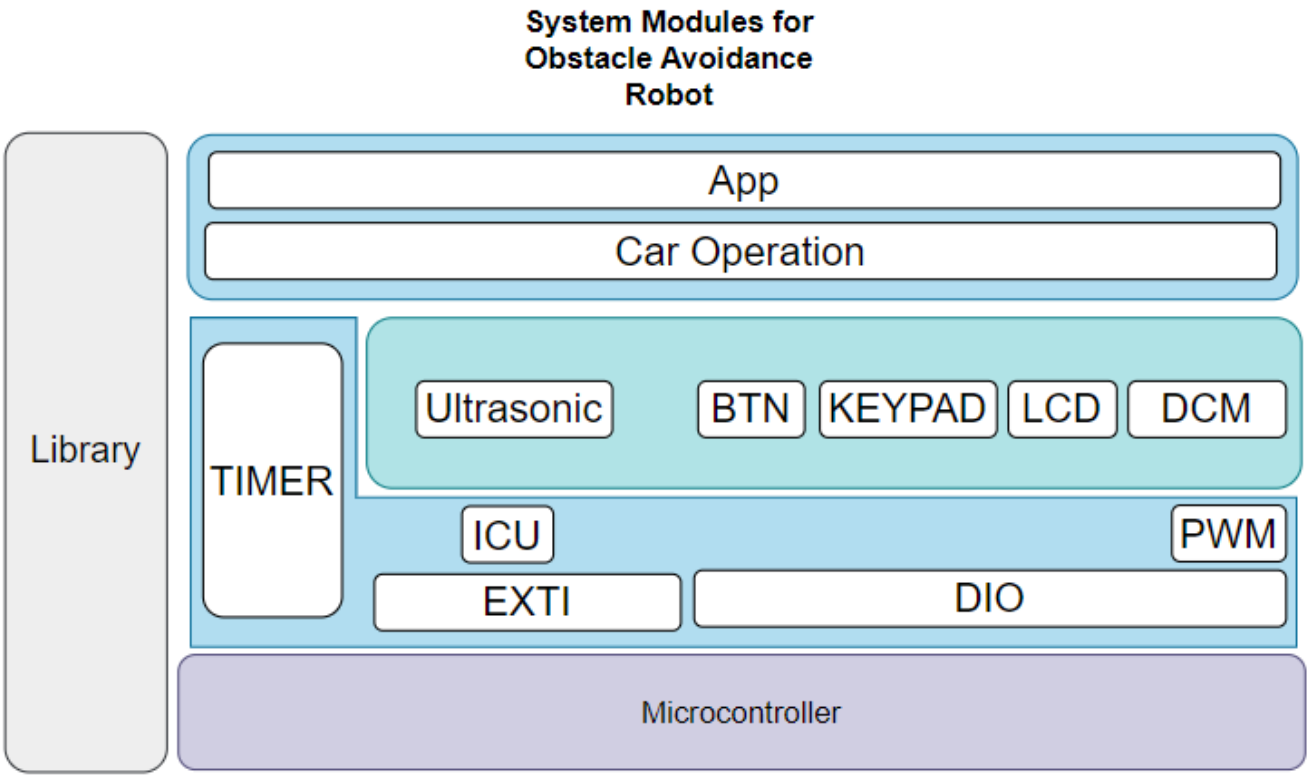
2.2.9. Keypad Module.

- A keypad module is a set of buttons arranged in a block or “pad” which bear digits, symbols or alphabetical letters. Pads mostly containing numbers are called a numeric keypad.

2.2.10. DC Motor Module.

- DC Motor is a device which transforms electrical energy into mechanical energy. Specifically, a DC motor uses DC current to convert electrical energy into mechanical energy. The basic principle of a motor is the interaction between the magnetic field and current to produce a force within the motor which helps the motor to rotate. So when the electric current is passed through a coil in a magnetic field, a magnetic force is generated which produces a torque resulting in the movement of the motor. The direction of the motor is controlled by reversing the current .

2.2.11. SystemModules.



2.3. APIs

2.3.1. Definition

- An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behaviour and capabilities of the component and its inputs and outputs.

2.3.2. MCAL_APIs

2.3.2.1. DIO

```
/*  Writes a voltage level to a pin of the DIO interface.
* Parameters|
*  [in] pin The pin to write the voltage level to
*  [in] volt The voltage level to write (HIGH or LOW
* Returns
*
*pin represents the pin to write the voltage level to, and volt represents the voltage
*level to write (HIGH or LOW).
*
*
* Parameters|        [in] pin The pin to write the voltage level to
*                    [in] volt The voltage level to write (HIGH or LOW).
* Returns
*      none
*/
void DIO_writepinn  (DIO_Pin_type  pin,DIO_PinVoltage_type volt)


/*  Reads the voltage level from a pin of the DIO interface.
*
*pin represents the pin to read the voltage level from, and volt is a pointer to store
*the read voltage level. The function reads the voltage level from the specified pin
*and stores it in the memory location pointed to by volt.
*
*
* Parameters|        [in] pin The pin to read the voltage level from.
*                    [out] volt Pointer to store the read voltage level.
* Returns
*      none
*/

void DIO_readpinn  (DIO_Pin_type  pin,DIO_PinVoltage_type *volt)
/*  Reads the voltage level from a pin of the DIO interface.
*
*pin represents the pin to read the voltage level from, and volt is a pointer to store
*the read voltage level. The function reads the voltage level from the specified pin
*and stores it in the memory location pointed to by volt.
*
*
* Parameters|        [in] pin The pin to toggle.
*
* Returns
*     none
*/
void DIO_togllepin (DIO_Pin_type  pin)

/*  Initializes a pin of the DIO interface with a given status.
*
*The function DIO_initpinn initializes a pin of the DIO interface with a given status.
*It takes two parameters: pin, which represents the pin number, and status, which
*represents the desired status of the pin (OUTPUT, INFREE, or INPULL).
*
```

```
 * Parameters
 *        [in] pin The The pin number.
 *        [in] status The status of the pin (OUTPUT, INFREE, or INPULL).
 *
 * Returns
 *      none
 */
void DIO_initpinn  (DIO_Pin_type  pin,DIO_PinStatus_type status)
```

2.3.2.2. EXI

```
/*
*Function: EXT_vINTERRUPT_Init
*
*Description: Initialises an external interrupt on a micro-controller with the specified
*configuration settings.
*
*Parameters: Void
*
*Return Type: Void
*
*Overall, the EXT_vINTERRUPT_Init function provides a way to Initialise an external interrupt
*on a micro-controller with the desired configuration settings. By using this function, the
*software can setup and handle external interrupts based on the specific interrupt number and
*sense control mode, and execute the appropriate ISR when the interrupt is triggered.
*/

void EXT_vINTERRUPT_Init(void);

/*
*Function: EXT_vINTERRUPT_Denit
*
*Description: Deinitialise an external interrupt on a microcontroller with the specified
*configuration settings.
*
*Parameters: Void
*
*settings for the external interrupt.
*
*Return Type: Void
*
*Overall, the EXT_vINTERRUPT_Denit function provides a way to deinitialize an external
*interrupt on a micro-controller with the desired configuration settings. By using this
*function, the software can remove the interrupt and associated ISR, freeing up resources
*and ensuring proper operation of the microcontroller.
```

```
*/

void EXT_vINTERRUPT_Denit(void);

/*
*Function Name : EXT_vINTERRUPT_setSenseControl
*
*Description: Configures the sense control of an external interrupt to determine its
*triggering conditions.
*
*Parameters: Interrupt number and sense control settings.
*
*Return Type: void.
*/

void EXT_vINTERRUPT_setSenseControl(void);

/*
*Function Name: EXT_INTERRUPT_SetInterruptHandler
*
*Description: Sets the interrupt handler function to be executed when an external interrupt
*occurs.
*
*Parameters: The interrupt handler function to be executed when the external interrupt occurs.
*
*Return Type: void
*/

void EXT_INTERRUPT_SetInterruptHandler(void);

/*
*Function Name: EXT0_INTERRUPT_SetInterruptHandler
*
*Description: Sets the interrupt handler function to be executed when an external interrupt
*occurs on EXT0 pin.
*
*Parameters: A function pointer to the interrupt handler function to be executed when the
*external interrupt occurs on the EXT0 pin. The function should take no arguments and return
*nothing.
*
*Return Type: Void.
*/

static void EXT0_INTERRUPT_SetInterruptHandler(void(*InterruptHandler)(void));



/*
*Function Name: EXT1_INTERRUPT_SetInterruptHandler
*
*Description: Sets the interrupt handler function to be executed when an external interrupt
*occurs on EXT1 pin.
```

```
*
*Parameters: A function pointer to the interrupt handler function to be executed when the
*external interrupt occurs on the EXT1 pin. The function should take no arguments and return
*nothing.
*
*Return Type: Void.
*/

static void EXT1_INTERRUPT_SetInterruptHandler(void(*InterruptHandler)(void));

/*
*Function Name: EXT2_INTERRUPT_SetInterruptHandler
*
*Description: Sets the interrupt handler function to be executed when an external interrupt
*occurs on EXT2 pin.
*
*Parameters: A function pointer to the interrupt handler function to be executed when the
*external interrupt occurs on the EXT2 pin. The function should take no arguments and return
*nothing.
*
*Return Type: Void.
*/

static void EXT2_INTERRUPT_SetInterruptHandler(void(*InterruptHandler)(void));
```

2.3.2.3. Timer

```
/* Initialises timer2 at normal mode
*
* This function initialises/selects the timer_2 normal mode for the
*  timer, and enables the ISR for this timer.
* Parameters
*            [in] en_a_interrputEnable value to set the interrupt
*                                      bit for timer_2 in the TIMSK reg.
* Return
*            An EN_TIMER_ERROR_T value indicating the success or failure of
*            the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
*            otherwise)
*/
EN_TIMER_ERROR_T TIMER_timer2NormalModeInit(EN_TIMER_ERROR_T   en_a_interrputEnable);
```

```
| Creates a delay using timer_2 in overflow mode
|
| This function Creates the desired delay on timer_2 normal mode.
| Parameters
|            [in] u16_a_interval value to set the desired delay.
| Return
```

```
|             An EN_TIMER_ERROR_T value indicating the success or failure of
|             the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
|             otherwise)
|
EN_TIMER_ERROR_T TIMER_intDelay_ms(Uint16_t  u16_a_interval);



| Start the timer by setting the desired prescaler.
|
| This function sets the prescaler for timer_2.
| Parameters
|             [in] u16_a_prescaler value to set the desired prescaler.
| Return
|             An EN_TIMER_ERROR_T value indicating the success or failure of
|             the operation
|              (TIMER_OK if the operation succeeded, TIMER_ERROR otherwise)
|
EN_TIMER_ERROR_T TIMER_timer2Start(Uint16_t u16_a_prescaler);

| Stop the timer by setting the prescaler to be 000--> timer is stopped.
|
| This function clears the prescaler for timer_2.
|
| Return
|      void
|
void TIMER_timer2Stop(void);
```

2.3.2.4. ICU

```
/* Initialises ICU_risingCaptureEdge at Rising Edge
*
* This function initialises/selects the timer_1 normal mode for the
*  timer, and enables the ISR for this timer.
* Parameters
*             [in] en_a_interrputEnable value to set the interrupt
*                                      bit for timer_0 in the TIMSK reg.
* Return
*             void
*/
void ICU_risingEdgeCapture(void);



/*
* Creates a delay using ICU_getVal
*
* This function Creates the desired delay on timer_2 normal mode.
* Parameters
*             [in] *u16_a_interval pointer to variable to store the ICU value.
* Return
*             void
```

```
*/
voidICU_getValue(Uint16_t  *u16_a_icuVal);


/*
* Start the timer by setting the desired prescaler.
*
* This function sets the prescaler for timer_2.
* Parameters
*            [in] u16_a_prescaler value to set the desired prescaler.
* Return
*            void
*/
void EXT_enablePIE(Uchart8_t u8_a_interrupted, Uchart8_t u8_a_senseControl);
/* Initialises timer1 at normal mode
*
* This function initialises/selects the timer_1 normal mode for the
*  timer, and enables the ISR for this timer.
* Parameters
*            [in] en_a_interrputEnable value to set the interrupt
*                                      bit for timer_1 in the TIMSK reg.
* Return
*            An EN_TIMER_ERROR_T value indicating the success or failure of
*            the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
*            otherwise)
*/
EN_TIMER_ERROR_T TIMER_timer1NormalModeInit(EN_TIMER_ERROR_T   en_a_interrputEnable);


/* Creates a delay using timer_1 in overflow mode
*
* This function Creates the desired delay on timer_1 normal mode.
* Parameters
*            [in] u16_a_interval value to set the desired delay.
* Return
*            An EN_TIMER_ERROR_T value indicating the success or failure of
*            the operation (TIMER_OK if the operation succeeded, TIMER_ERROR
*            otherwise)
*/
EN_TIMER_ERROR_T TIMER_delay_ms(Uint16_t  u16_a_interval);



/* Start the timer by setting the desired prescaler.
*
* This function sets the prescaler for timer_1.
* Parameters
*            [in] u16_a_prescaler value to set the desired prescaler.
* Return
*            An EN_TIMER_ERROR_T value indicating the success or failure of
*            the operation
*             (TIMER_OK if the operation succeeded, TIMER_ERROR otherwise)
```

```
*/
EN_TIMER_ERROR_T TIMER_timer1Start(Uint16_t u16_a_prescaler);

/* Stop the timer by setting the prescaler to be 000--> timer is stopped.
*
* This function clears the prescaler for timer_1.
*
* Return
*      void
*/
void TIMER_timer1Stop(void);
```

2.3.2.5. PWM

```
/*
* Function    : TIMER0_init
* Description : this function initializes timer0 with normal mode Also enable peripheral and
                Global interrupt.
* Args        : void
* return      : void
*/
void TIMER0_init(void);

/*
* Function    : TIMER0_start
* Description : this function set three clock bits with chosen prescaller in config file
                (timer starts when we call this function)
* Args        : void
* return      : void
*/
void TIMER0_start(void);

/*
* Function    : TIMER0_stop
* Description : this function clear three clock bits (timer stops when we call this function)
* Args        : void
* return      : void
*/
void TIMER0_stop(void);

/*
* Function    : TIMER0_initPWM
* Description : this function initializes all pwm pins as outputs and set high on them,
                also calls TIMER0_init ....
* Args        : void
* return      : void
*/
void TIMER0_initPWM(void);
/*
* Function    : TIMER0_initPWM
* Description : this function calculates onTime and offTime , also calls TIMER0_start ....
* Args        : DutyCycle (0--->100)
* return      : void
```

```
*/
void TIMER0_initPWM(void);


/*
* Function    : TIMER0_PWM_ExecutedFunction
* Description : this function switches level of cycle based on global on_off_state
                (this function called from ISR)
* Args        : void
* return      : void
*/
static void TIMER0_PWM_ExecutedFunction(void);
```

2.3.3. ECUAL_APIs

2.3.3.1. Push-BUTTON

```
/*
*Function: PUSH_BTN_intialize
*
*Description: Initialises a push button Pins based on the configuration settings.
*
*Parameters: Void.
*
*Return Type: Void.
*/

void PUSH_BTN_intialize(void);



/*
*Function     : PUSH_BTN_read_state
*
*Description  : Reads the current en_g_state of a push button and returns its value.
*
*Parameters:
*btn          : A pointer to an ST_PUSH_BTN_t struct that contains the configuration settings
*and current en_g_state information for the push button.
*btn_state : A pointer to an EN_PUSH_BTN_state_t enum where the current en_g_state of the push
*button will be stored.
*Return Type  : Void.
*
*Overall, the PUSH_BTN_read_state function provides a way to read the current en_g_state of a
*push button and return its value. By using this function, the software can determine whether
*the push button is currently pressed or released and take appropriate action based on its
*en_g_state.
*/
void PUSH_BTN_read_state(Uchar8_t btnNumber , EN_PUSH_BTN_state_t);
```

2.3.3.2. Keypad

```
/*
* Function    : KEYPAD_init
* Description : Initializes Rows as Output and Cols as input and put high logic on each pin
* Args        : Void
* return      : Void
*/
void KEYPAD_init(void);

/*
*Function    : KEYPAD_getButton
Description : loops over rows and cols and return keyPressed or nothing pressed
Args        : Void
return      : EN_KEYPADKEYS enum holds all possible keys
 */
EN_KEYPAD_KEYS KEYPAD_getButton(void);
```

2.3.3.3. LCD

```
|  Writes an instruction to the DIO interface
|
|The WriteIns function is used to write an instruction to the DIO (Digital |Input/Output)
interface. It takes an 8-bit instruction (ins) and performs the |necessary DIO write
operations to send the instruction
|
|
| Parameters        [in] ins The instruction to write.
|
| Returns
|      none
|
void WriteIns(Uchar8_t ins);


|  Writes data to the DIO interface.
|
|The WriteData function is used to write data to the DIO (Digital Input/Output) |interface. It
takes an 8-bit data value (data) and performs the necessary DIO write |operations to send the
data.
|
|
| Parameters        [in] data The data to write.
|
| Returns
|      none
|
|
void WriteData(Uchar8_t data);
```

```
|Initialises the LCD display.
|
|The LCD_Init function is responsible for initialising the LCD display. It performs a |series
of command writes to set up the display parameters and configure its behaviour
|
|
| Parameters none
|
| Returns
|      none

void LCD_Init(void);

|  Writes a character to the LCD display.
|
|The LCD_WriteChar function takes a character ch as input and calls the WriteData |function to
write the character to the LCD display.
| Parameters
|        [in] ch The character to be written.
|       |
| Returns
|      none
|
void LCD_WriteData(Uchar8_t ch);

|  Writes a string of characters to the LCD display.
|
|The LCD_WriteString function takes a null-terminated string str as input and iterates
|through each character in the string.
| Parameters
|        [in] str The string to be written.
|       |
| Returns
|      none
|
void LCD_WriteString(Uchar8_t *str);

|  Sets the cursor position on the LCD display.
|
|The LCD_WriteChar function takes a character ch as input and calls the WriteData |function to
write the character to the LCD display.
| Parameters
|     [in]line The line number (0 or 1) where the cursor should be positioned.
|     [in]cell number (0 to 15) within the line where the cursor should be positioned.
| Returns
|      none
|
void LCD_SetCursor(Uchar8_t line,Uchar8_t cell);

| Clears the content of the LCD display.
|
|The LCD_Clear function sends the clear display instruction (CLR_INS) to the LCD |display
using the WriteIns function..
```

```
| Parameters
|         none
|
| Returns
|      none
|
void LCD_Clear(void);
```

```
| Writes a signed 32-bit integer to the LCD display.
|
|The LCD_WriteNumber function takes a signed 32-bit integer num as input and displays
|it on the LCD. It uses an array str to store the individual digits of the number in
|reverse order.
| Parameters
|        [in] num The number to be displayed.
|
| Returns
|      none
|
void LCD_WriteNumber(Sint32_t num);
```

```
| Initializes the pins of the LCD interface.
|
|The LCD_PinsInit function initializes the pins used for interfacing with the LCD. It
|calls the DIO_initpinn function to set the direction of each pin to output.
| Parameters
|         none
|
| Returns
|      none
void LCD_PinsInit();
```

2.3.3.4. DCM

```
/*
* DCM_motorInint
*
* This function initialises/selects the desired pins for the two motors
*  using the passed configuration
* Parameters
*            [in] *DCM_a_ptrToConfig pointer to the configuration struct
* Return
*            [out] EN_DCM_ERROR_s error statue for DC motor.
*/
EN_DCM_ERROR_T DCM_motorInit(ST_DCM_config_t *DCM_a_ptrToConfig);


/*
* DCM_changeDiretion
*
* This function changes the motor direction
* Parameters
*            [in] *DCM_a_ptrToConfig pointer to the configuration struct
*            [in]  DCM_a_motorNum enum selection for the motor side.
* Return
*            [out] EN_DCM_ERROR_s error statue for DC motor.
*/
EN_DCM_ERROR_T DCM_changeDiretion(ST_DCM_config_t *DCM_a_ptrToConfig, EN_DCM_MOTORSIDE
DCM_a_motorNum);


/*
* Stop the motor
*
* Parameters
*            [void
* Return
*            void
*/
void DCM_vdStopDCM(void);


/*
*
* This function set the desired PWM.
* Parameters
*            [in] DCM_a_dutyCycleValuevalue set the PWM.
* Return
*            [out] EN_DCM_ERROR_s error statue for DC motor.
*/
EN_TIMER_ERROR_T DCM_setDutyCycle(Uchart8_t DCM_a_dutyCycleValue);
```

```
/*
* This function rotates the desired motor.
* Parameters
*            [in] DCM_a_motorNum enum selection for the motor side.
*            [in] DCM_a_dutyCycleValuevalue set the PWM.
* Return
*            [out] EN_DCM_ERROR_s error statue for DC motor.
*/
EN_TIMER_ERROR_T DCM_rotateDCM(EN_DCM_MOTORSIDE DCM_a_motorNum,Uchart8_t
DCM_a_rotateSpeed );


/*
* Move the robot forward
*
* Parameters
*            [void]
* Return
*            [void]
*/
void DCM_MoveForward(void);

/*
* Move the robot backward
*
* Parameters
*            [void]
* Return
*            [void]
*/
void DCM_MoveBackward(void);
```

2.3.3.5 Ultrasonic

```
/*
*Function : ultrasonic_vInit
*
*Description: Initialises the ultrasonic sensor module for communication with the
*microcontroller.
*
*Parameters: None.
*
*Return Value: None.
*/
void ultrasonic_vIint(void);

/*
*Function : ultrasonic_vGetDistance
*
*Description: Measures the distance to an object using the ultrasonic sensor and returns the
```

```
*result through a pointer to a float variable.
*
*Parameters: A pointer to a float variable where the measured distance value will be stored.
*
*Return Value: Void.
*/
void ultrasonic_vGetDistance(float64_t *Copy_f64distance);
```

2.3.4. Application_APIs

2.3.4.1. Car_operations

```
/*
* Function    : setDefaultRotation
* Description: this function sets the rotation (right/left)
* Args        : void
* return      : void
*/

void setDefaultRotation(void);

/*
* Function    : obstacleMoreThan70
* Description: this function move the robot with 30 % speed and increases it’s speed to 50 %
               after 5 seconds if obstacles are far than 70 cm
* Args        : void
* return      : void
*/

void obstacleMoreThan70(void);

/*
* Function    : obstacleMoreThan30
* Description: this function move the robot with 30 % speed if obstacles are far than 30 cm
               and less than 70 cm
* Args        : void
* return      : void
*/
void obstacleMoreThan30(void);

/*
* Function    : obstacleMoreThan20
* Description: this function stops the robot and rotate 90 degree if it rotate 360 without
               finding any way more than 30 cm it stops 3 seconds then check again.
* Args        : void
* return      : void
*/
```

```
void obstacleMoreThan20(void);
/*
* Function    : obstaclelessThan20
* Description: this function stops the robot and move backward with 30 % speed
* Args        : void
* return      : void
*/
void obstacleLessThan20(void);

/*
* Function    : LCD_update
* Description: it updates lcd data
* Args        : speed,direction,distance
* return      : void
*/
void LCD_update(EN_SPEED u8_a_speed,EN_DIRECTION u8_a_direction,float64
f64_a_distance);

/*
* Function    : Car_Stop
* Description: Stop the Robot
* Args        : void
* return      : void
*/
void Car_Stop(void);
```

2.3.4.2. App

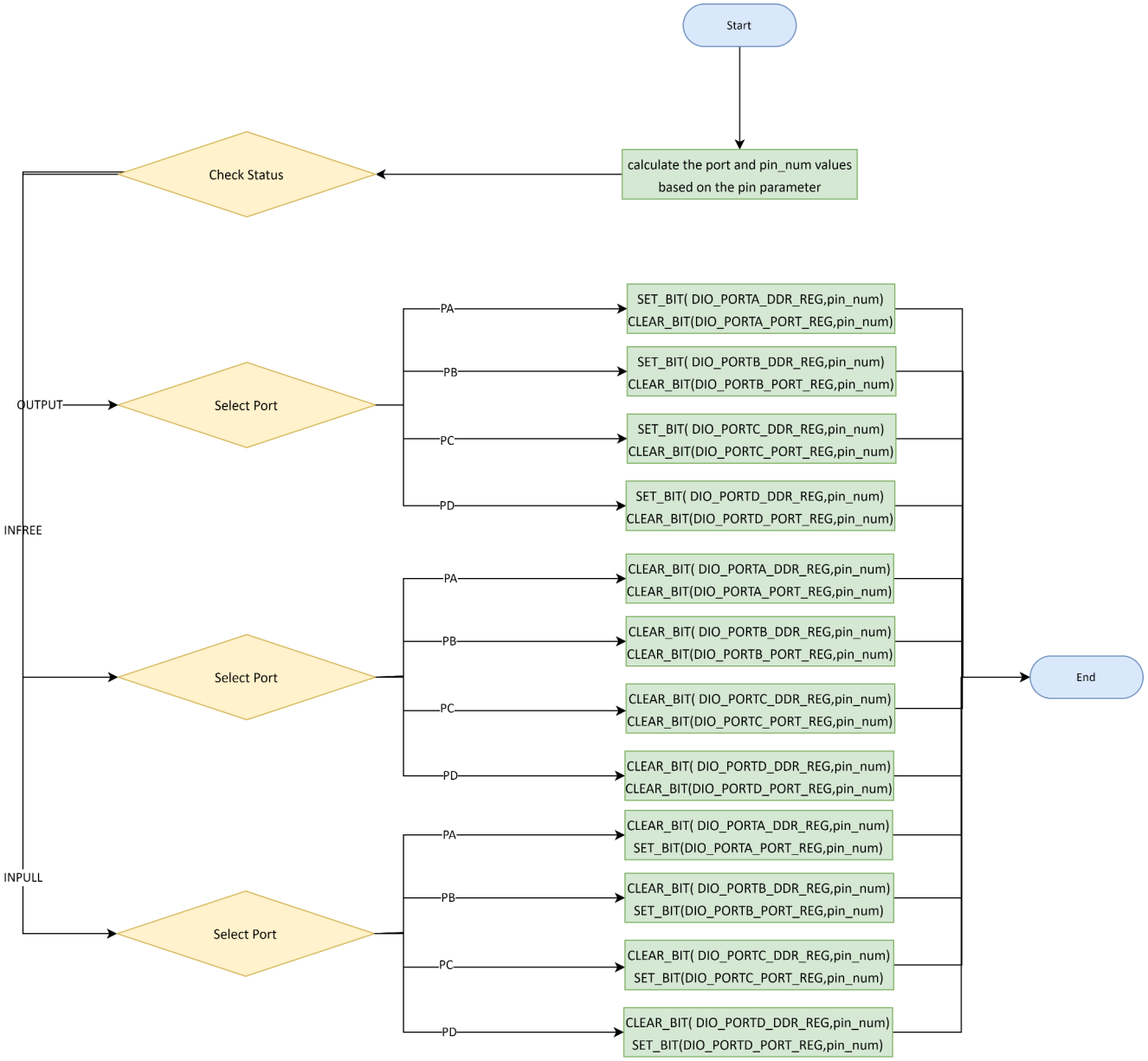
```
/*
* Function    : app_init
* Description: this function initialize all required drivers in hal layer
* Args        : void
* return      : void
*/

void app_init(void);

/*
* Function    : app_main
* Description: this function contain all app logic, must called in super loop
* Args        : void
* return      : void
*/

void app_main(void);
```

3.Low Level Design

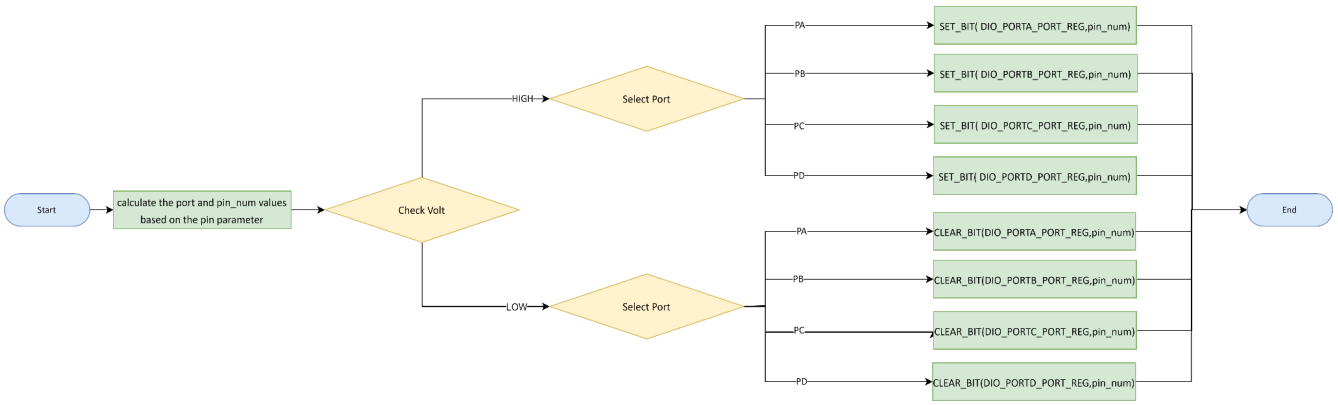
3.1. MCAL

3.1.1. DIO

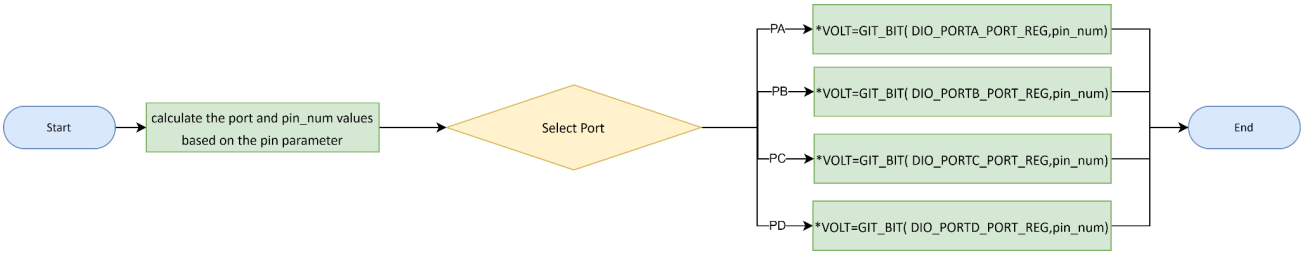
3.1.1.1. DIO_init



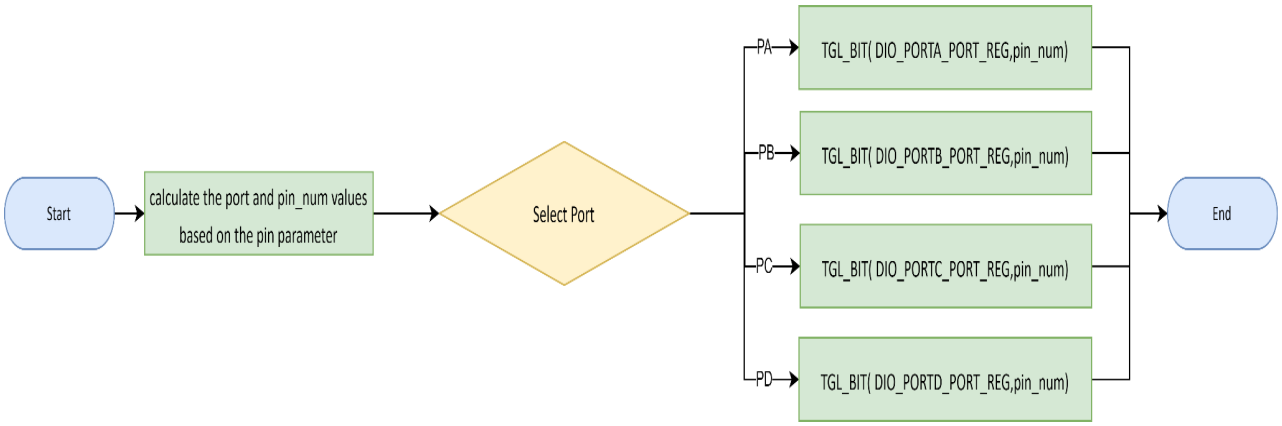
3.1.1.2. DIO_writepinn



3.1.1.3. DIO_readpin

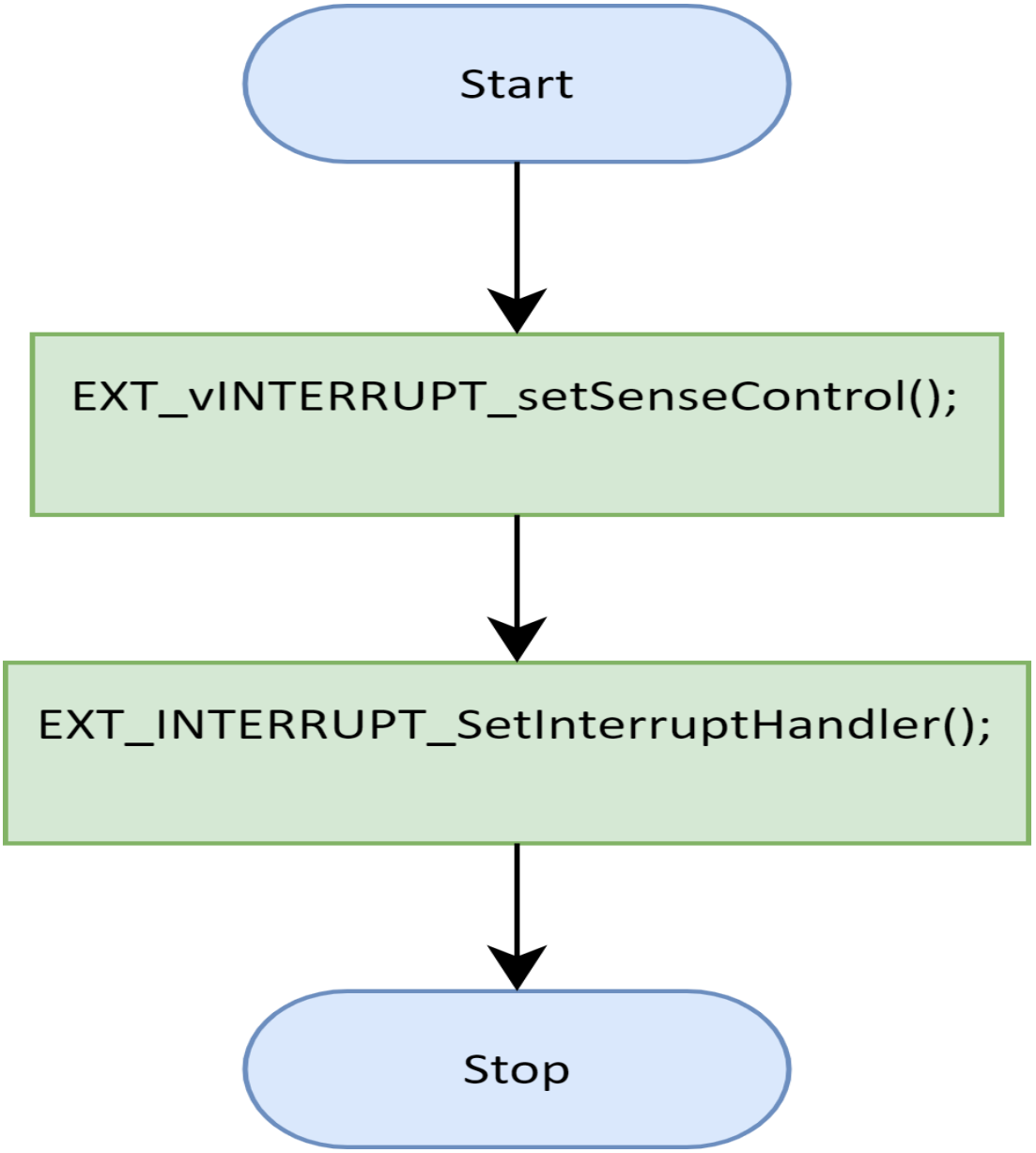


3.1.1.4. DIO_togglepin

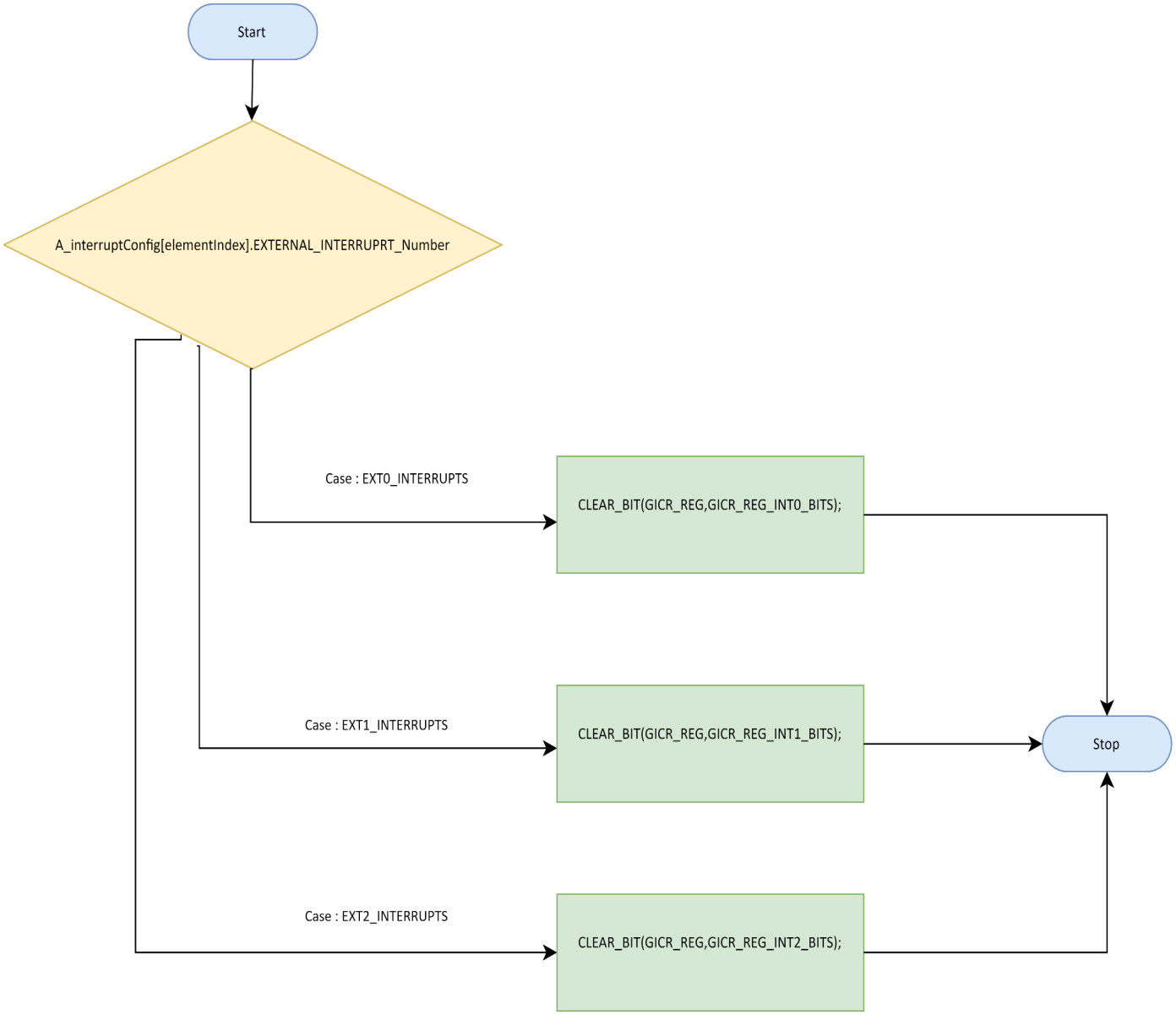


3.1.2. EXI

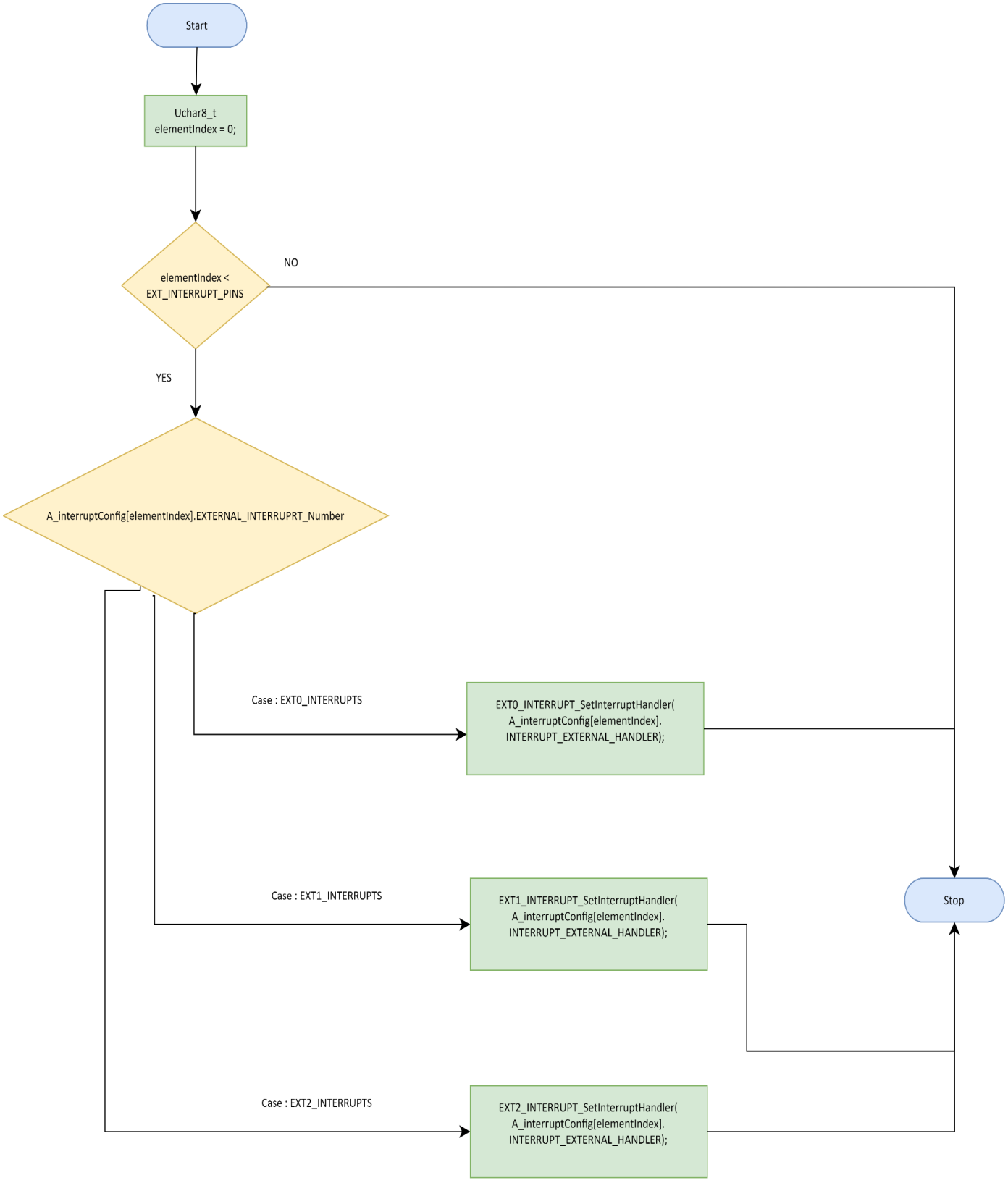
3.1.2.1 EXT_vINTERRUPT_Init



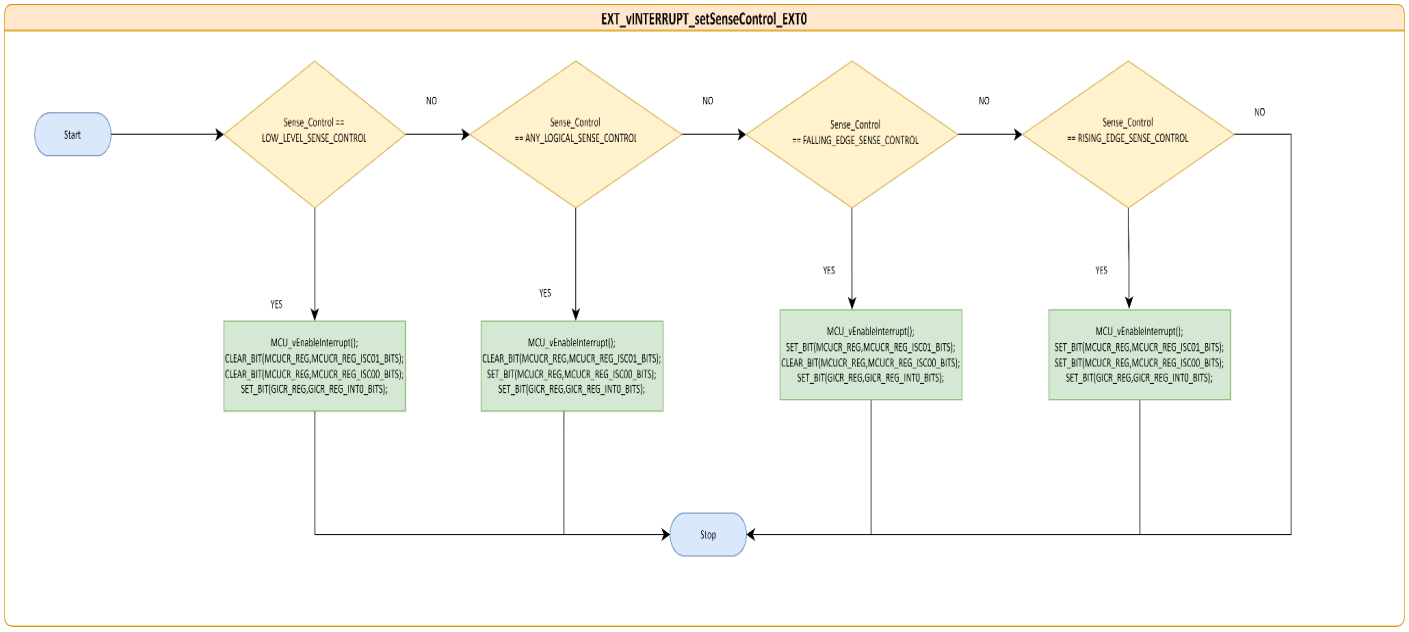
3.1.2.2 EXT_vINTERRUPT_Denit



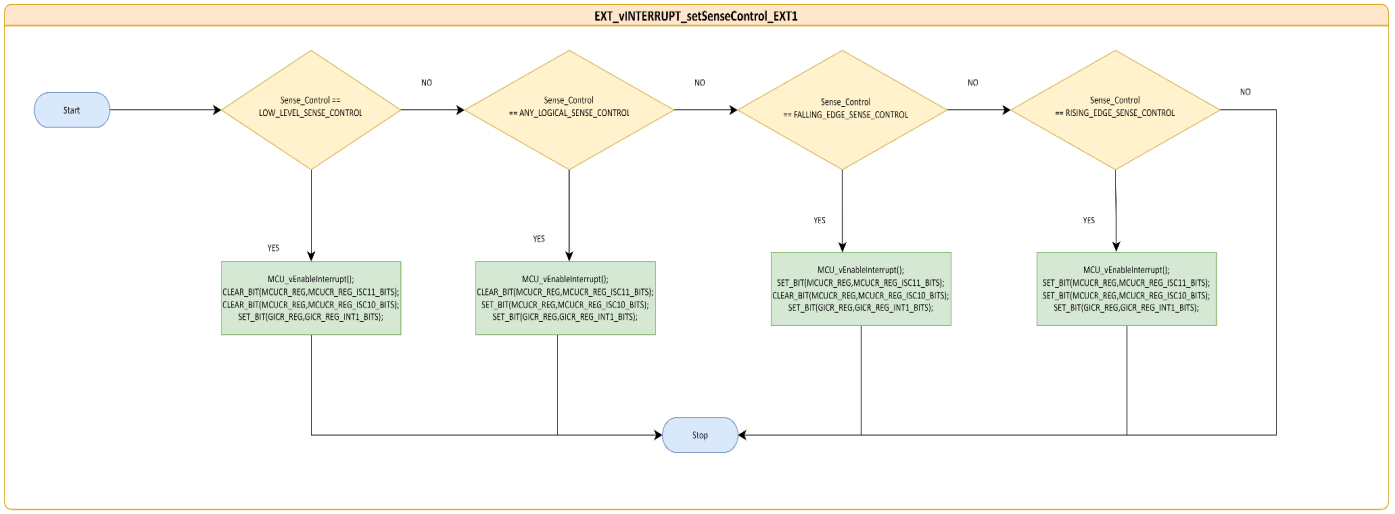
3.1.2.3 EXT_vINTERRUPT_setSenseControl



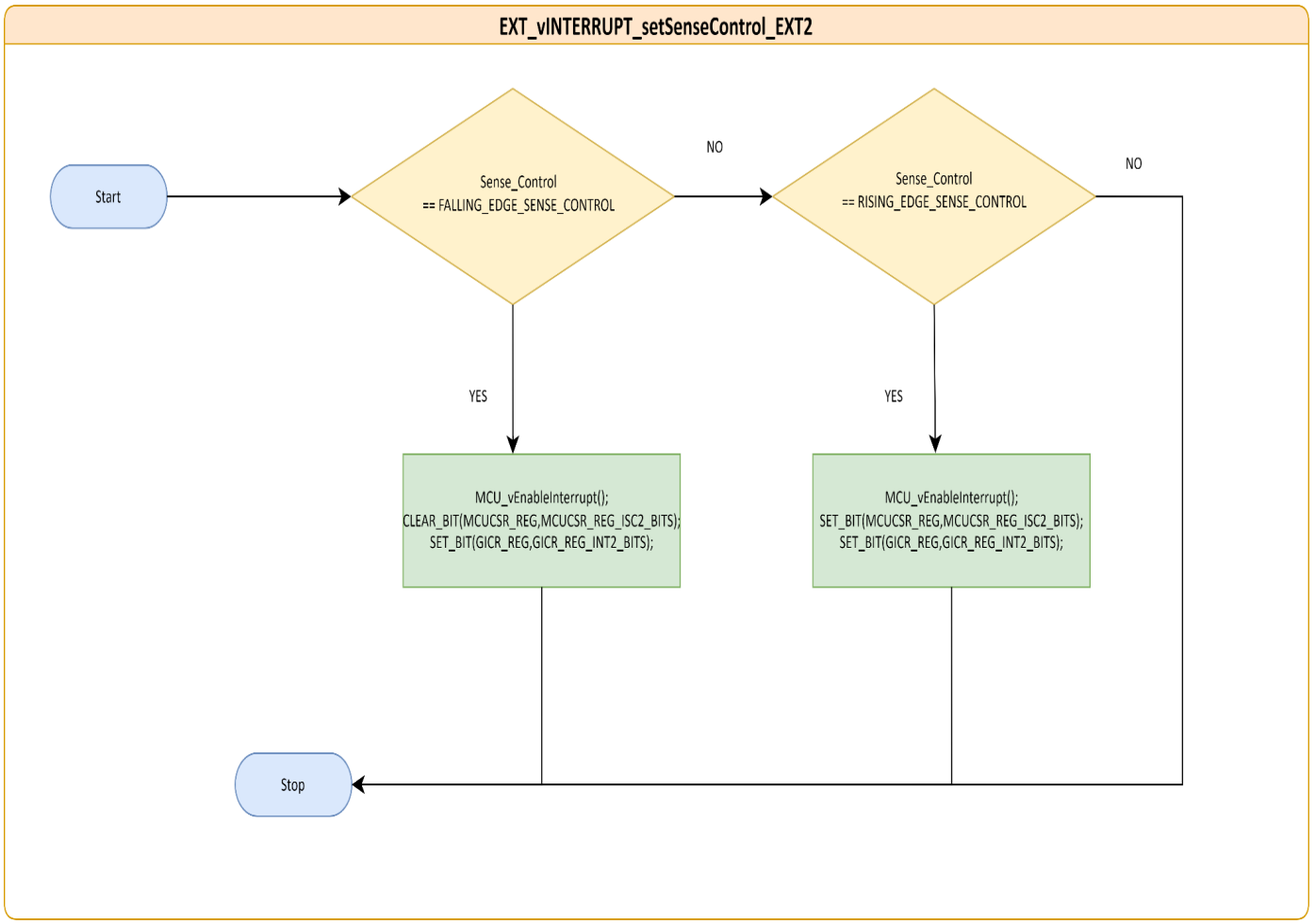
3.1.2.3.1 EXT_vINTERRUPT_setSenseControl_EXT0



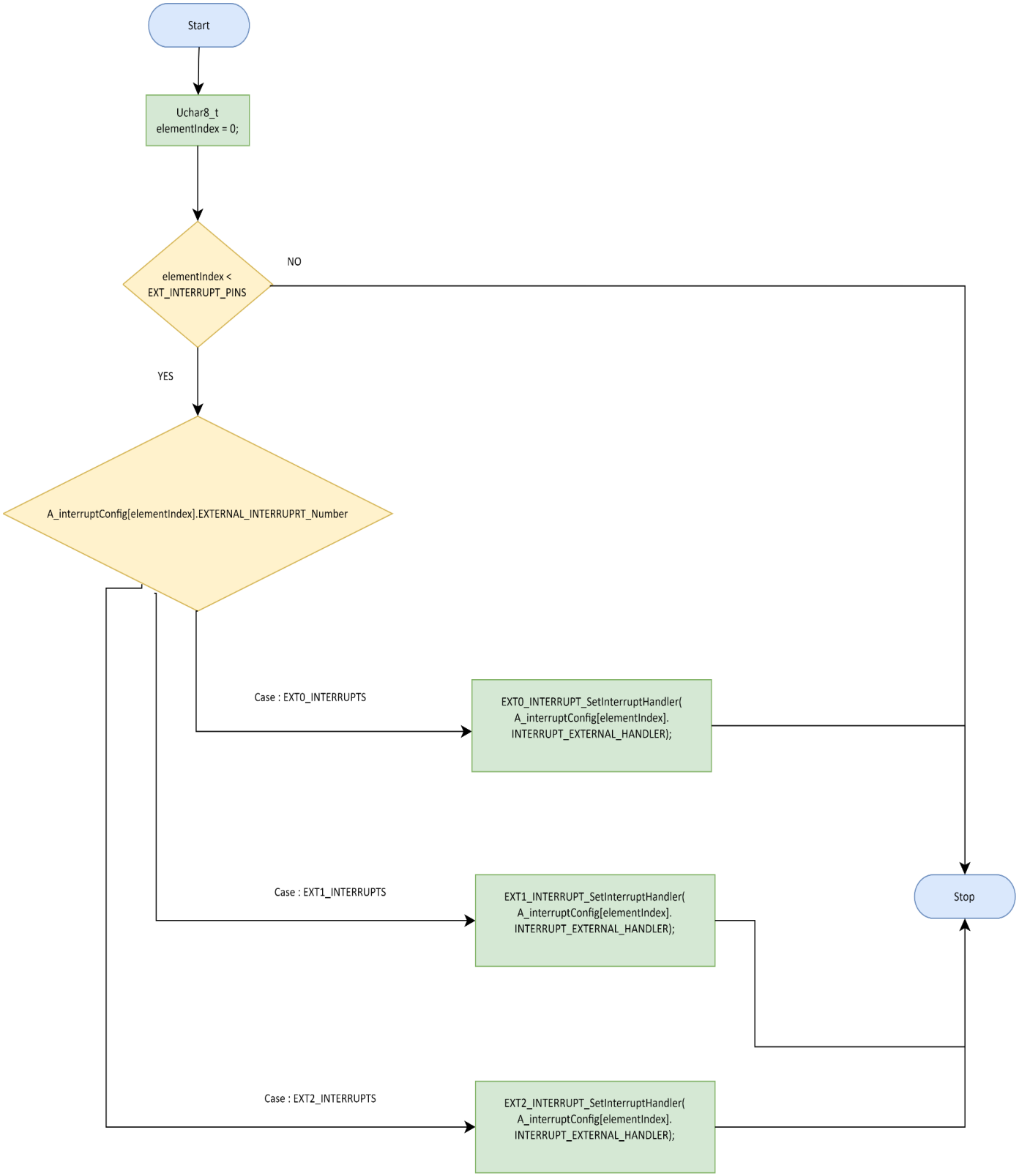
3.1.2.3.2 EXT_vINTERRUPT_setSenseControl_EXT1



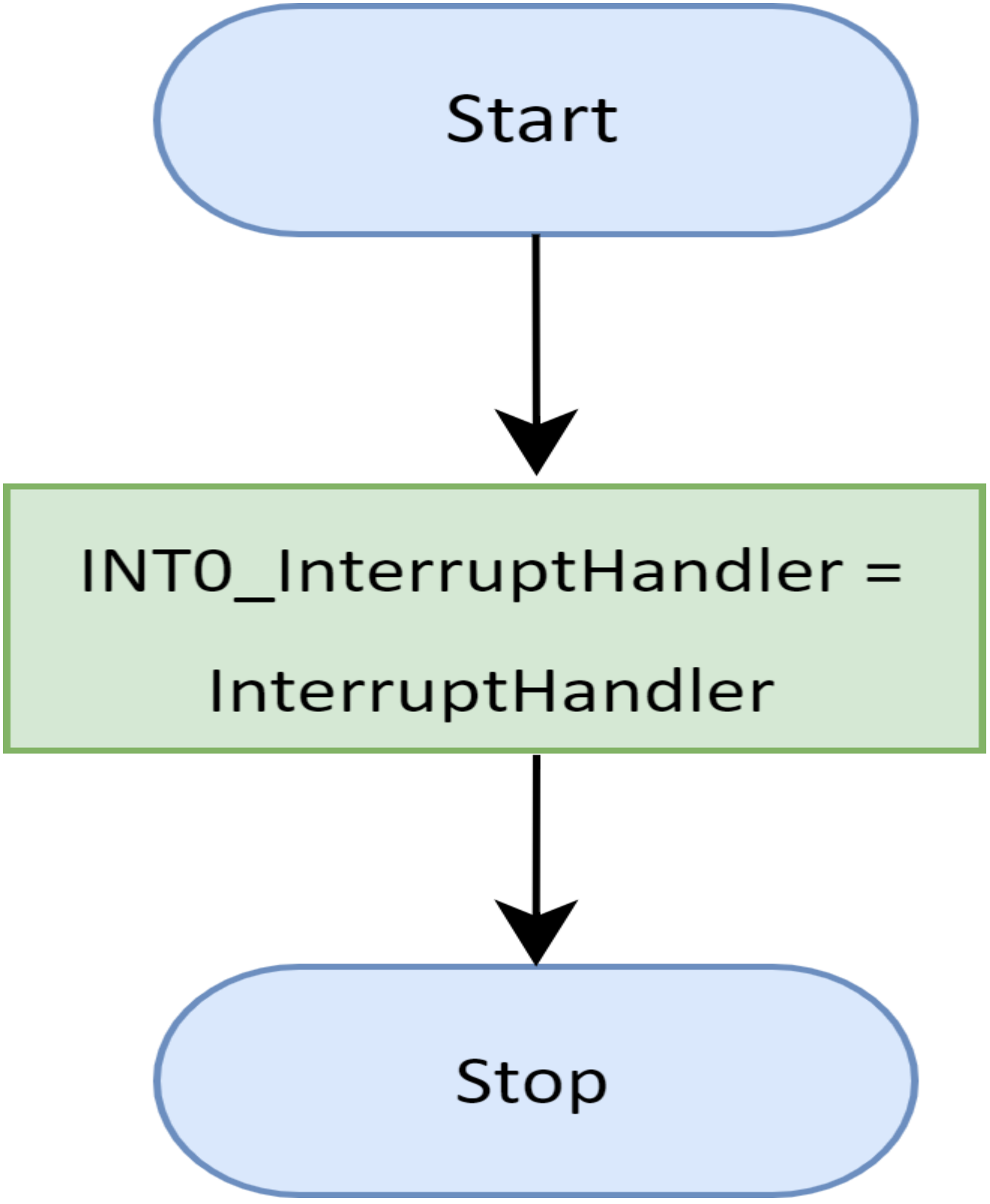
3.1.2.3.3 EXT_vINTERRUPT_setSenseControl_EXT2



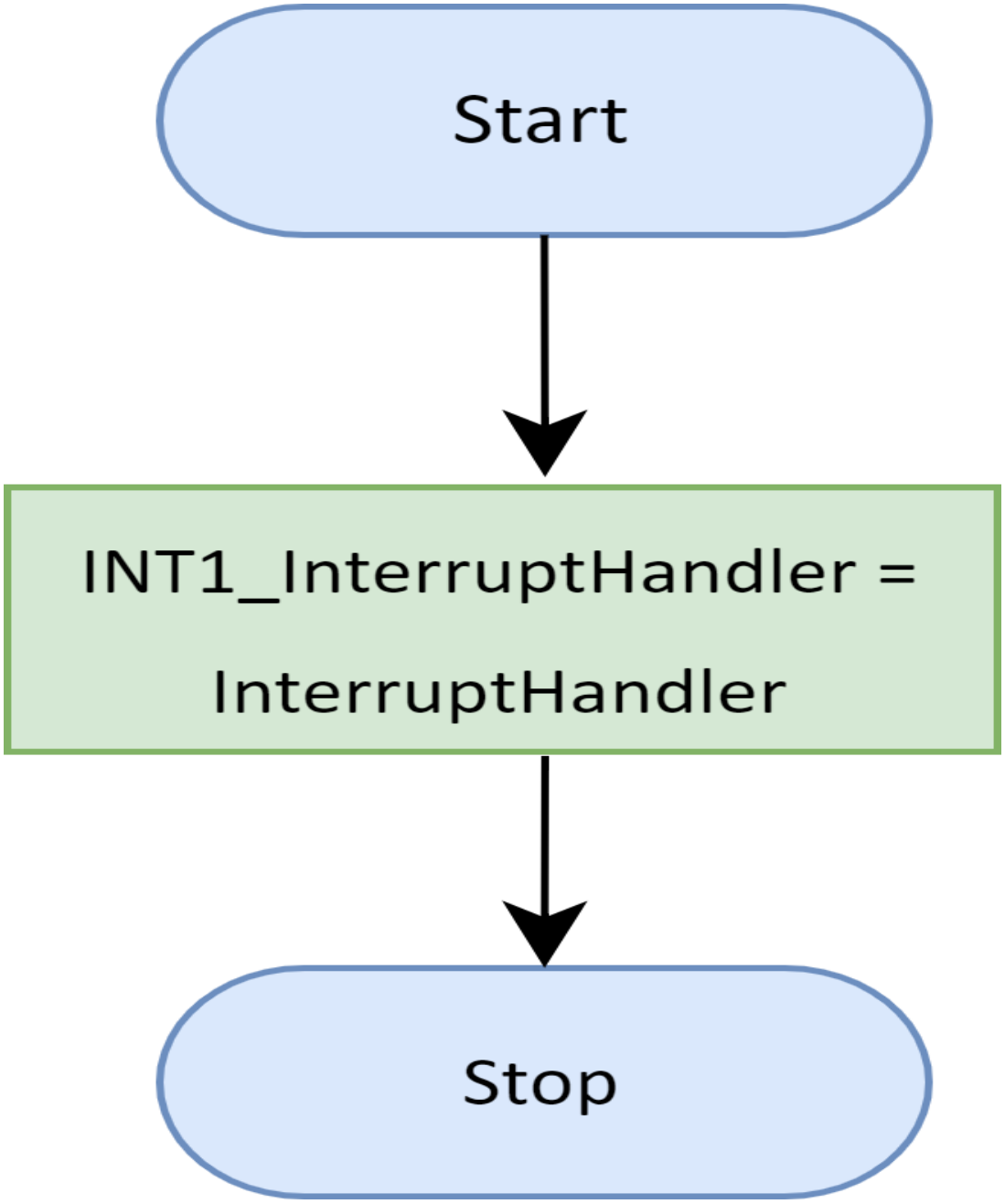
3.1.2.4 EXT_INTERRUPT_SetInterruptHandler



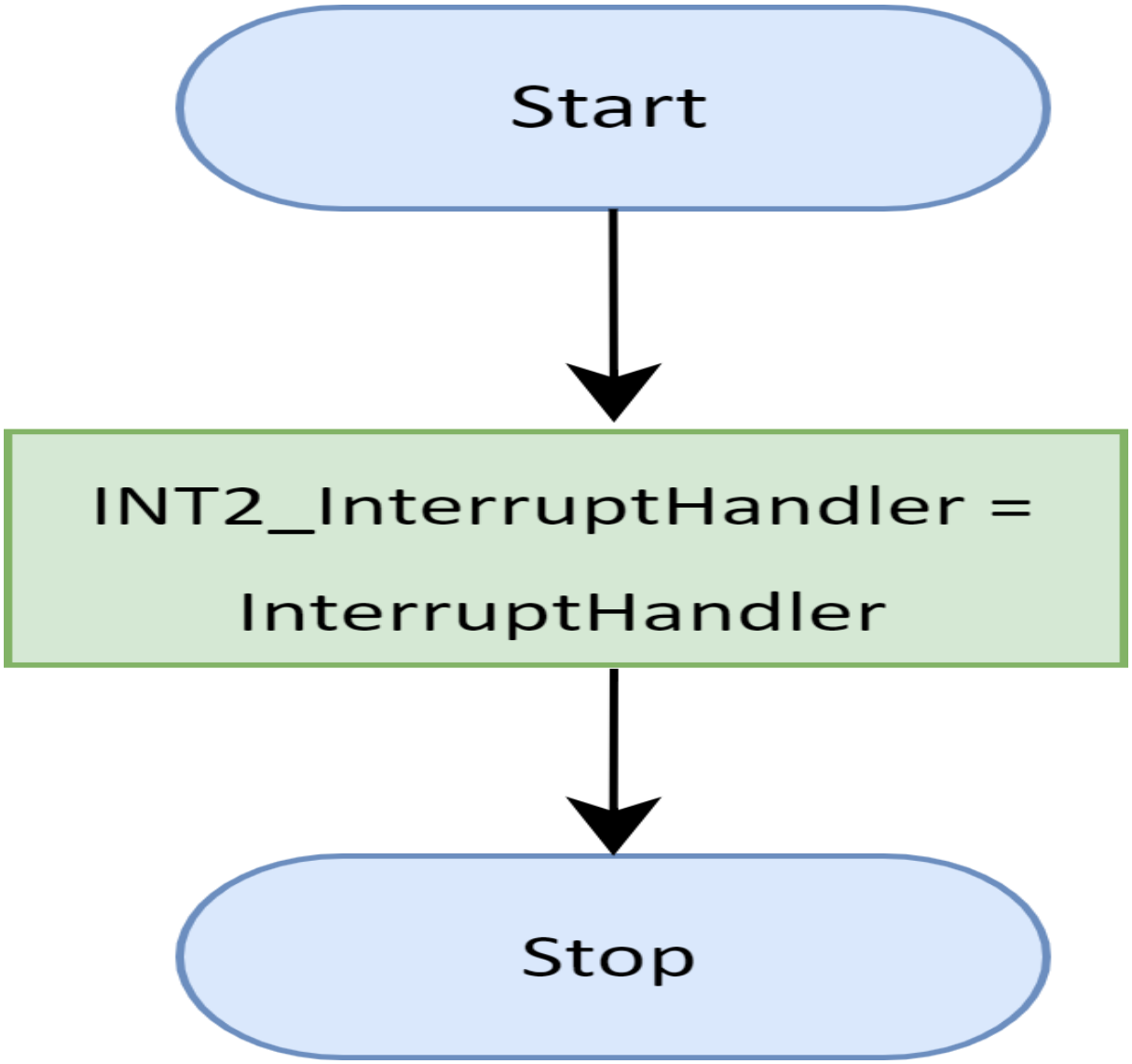
3.1.2.5 EXT0_INTERRUPT_SetInterruptHandler



3.1.2.6 EXT1_INTERRUPT_SetInterruptHandler

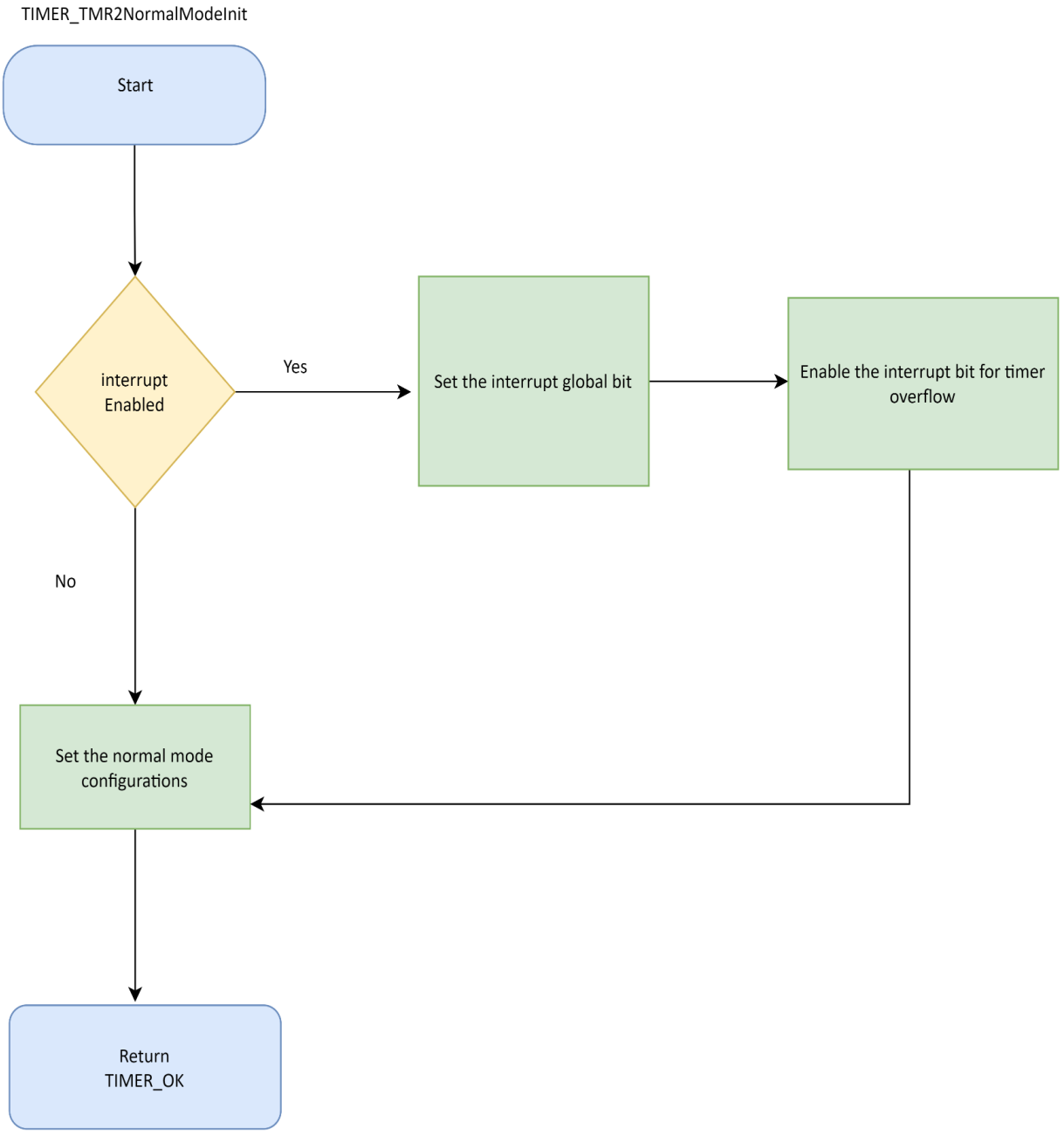


3.1.2.7 EXT2_INTERRUPT_SetInterruptHandler

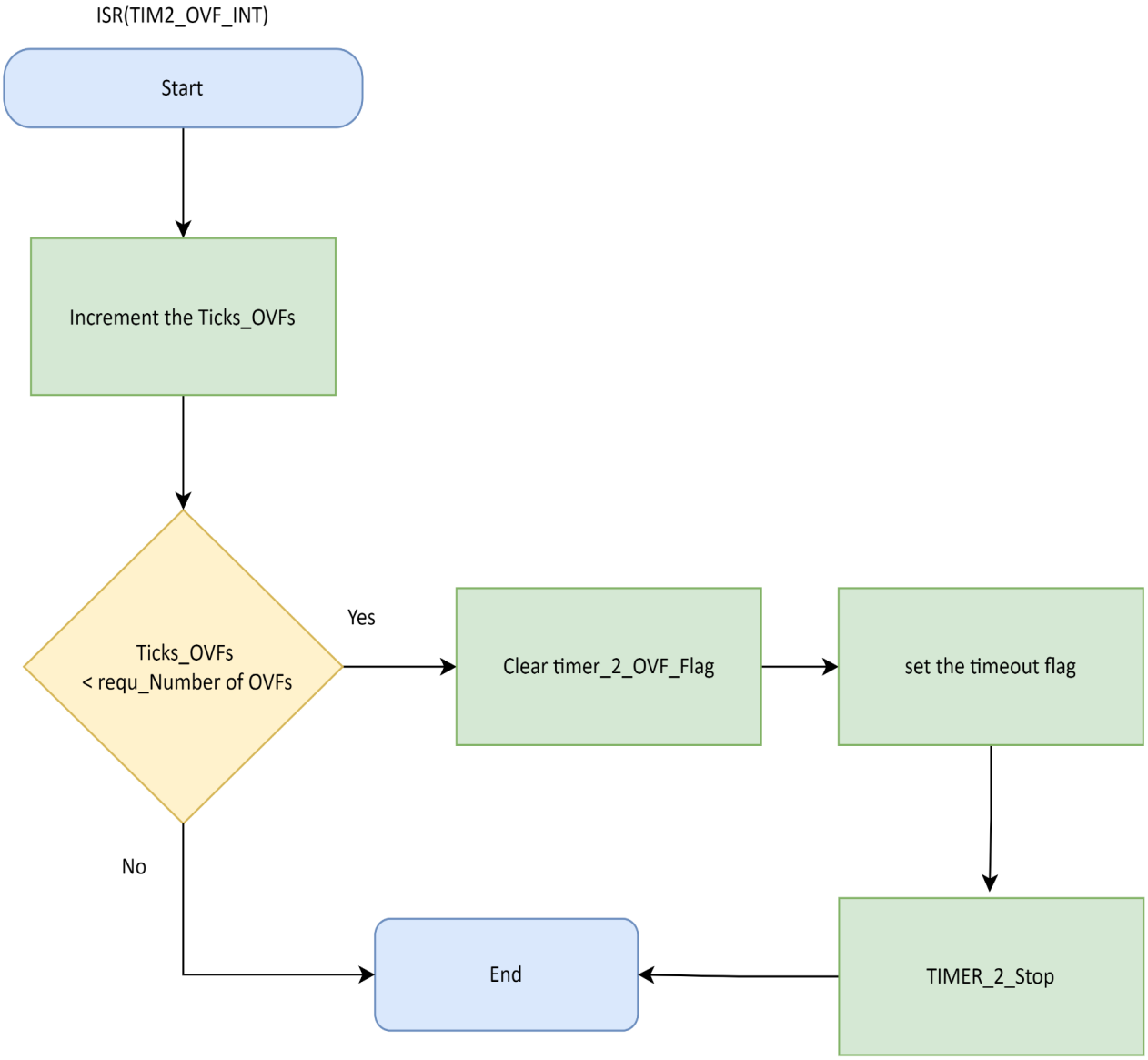


3.1.3. Timer

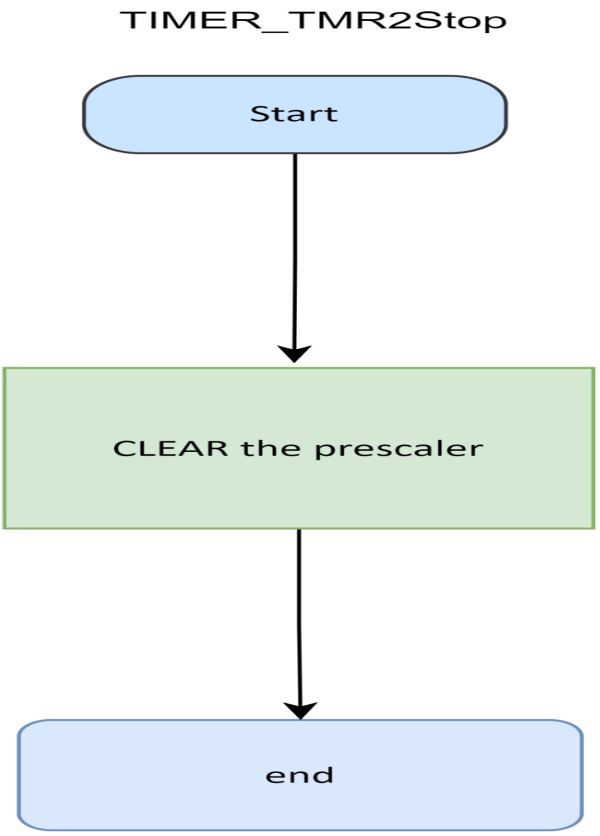
3.1.3.1. TIMER_TMR2NormalodeInit



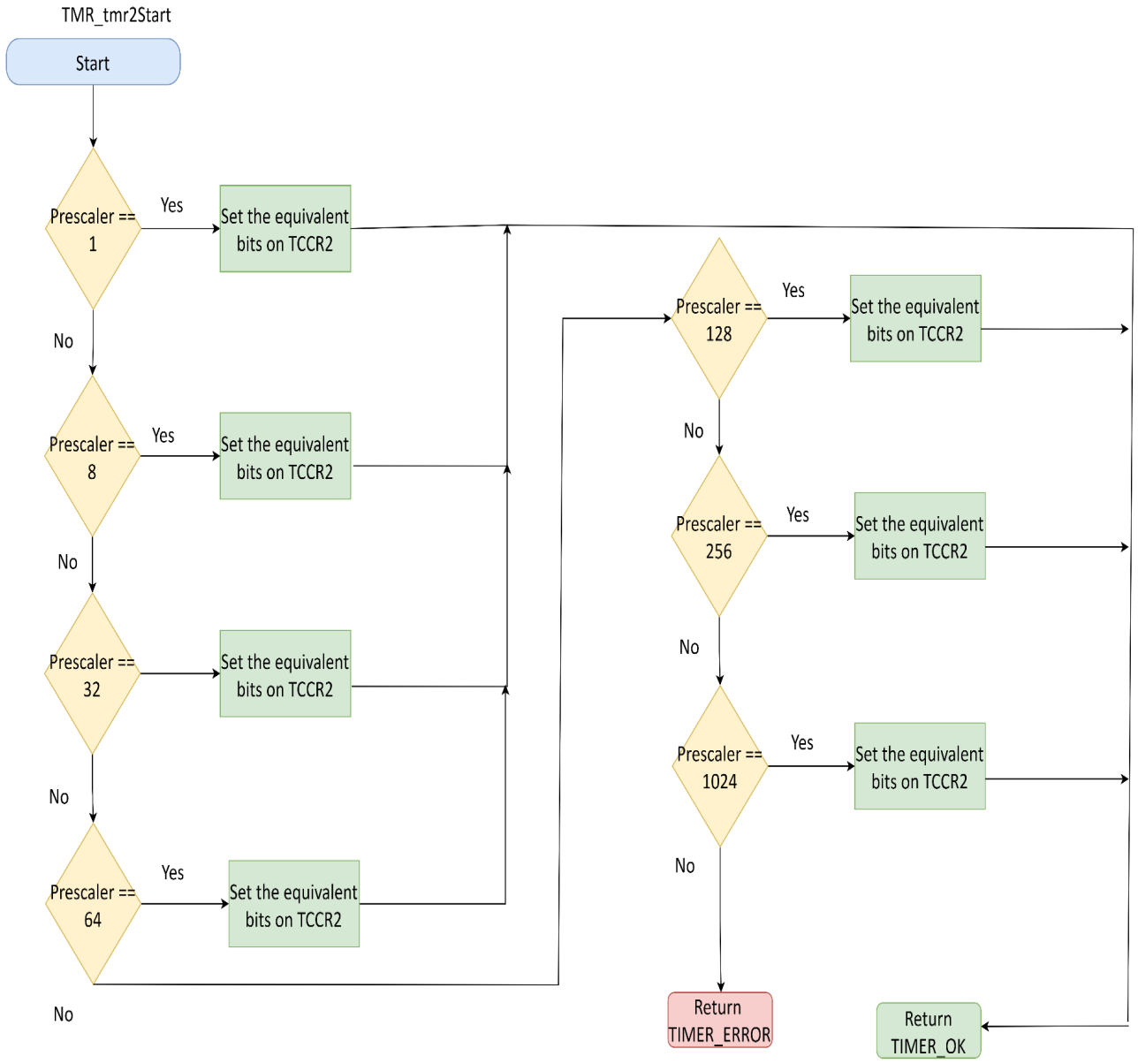
3.1.3.2. TIMER_TMR2ISR



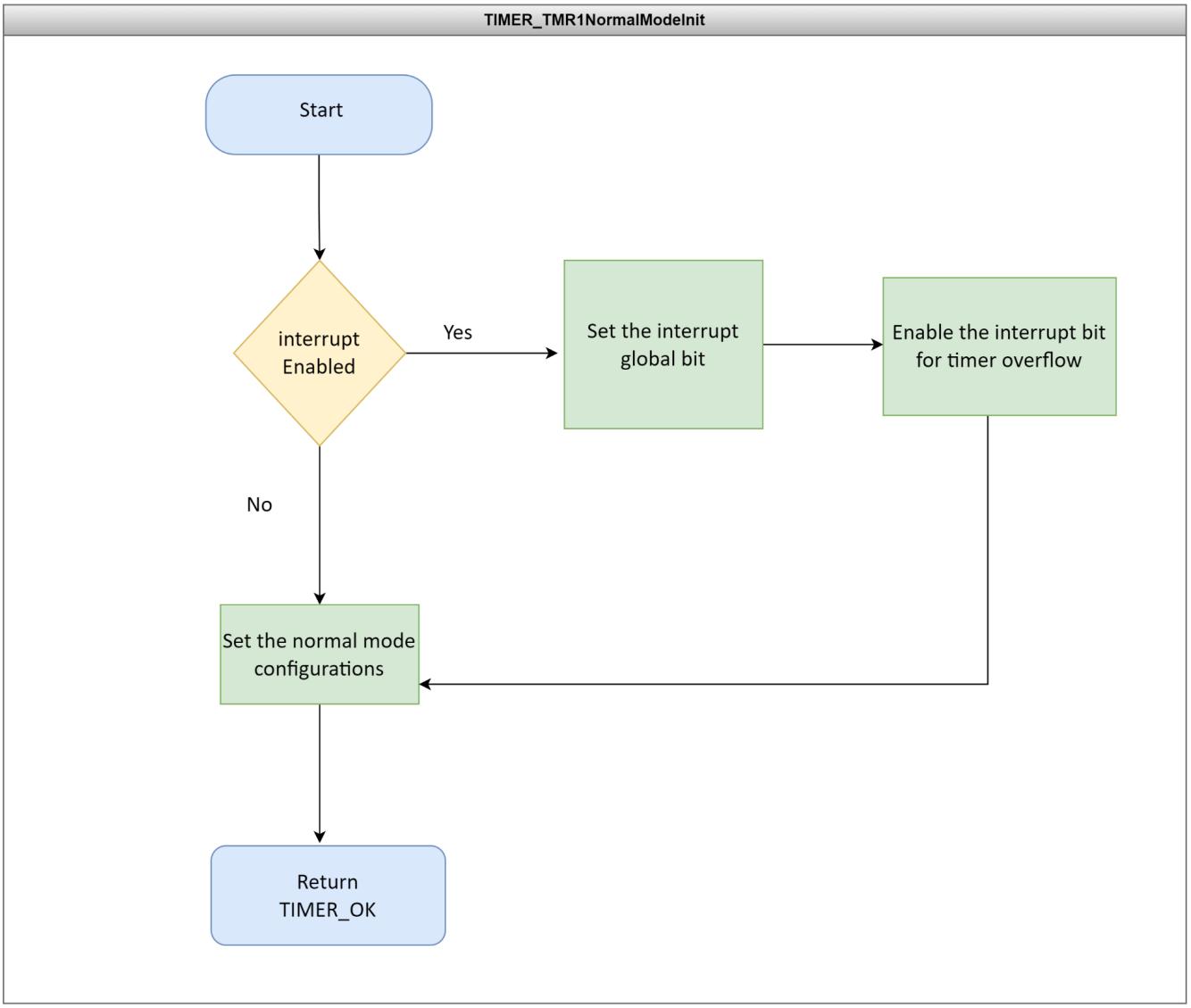
3.1.3.3 TIMER_vdStop



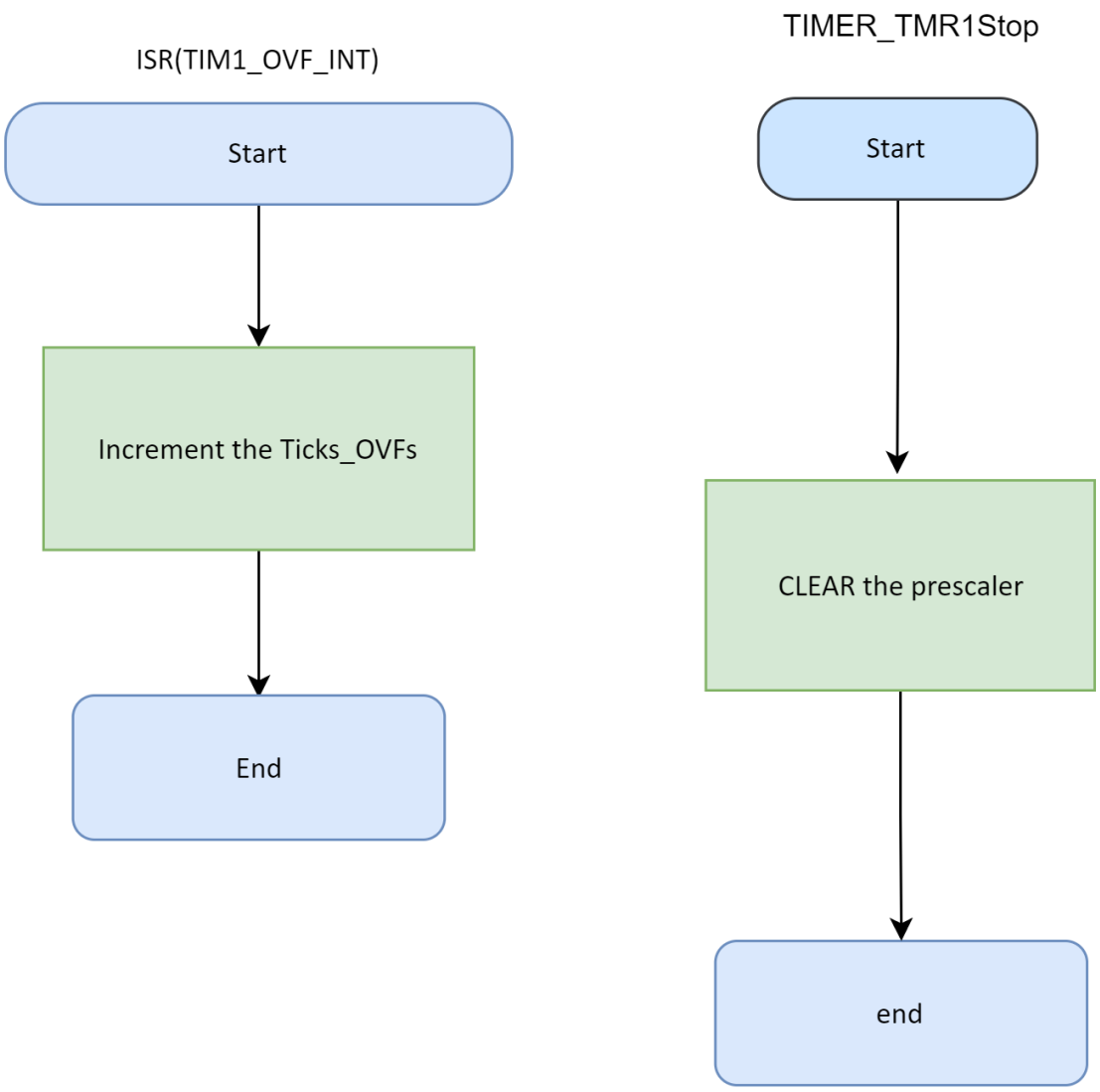
3.1.3.4. TIMER_TMR2Start



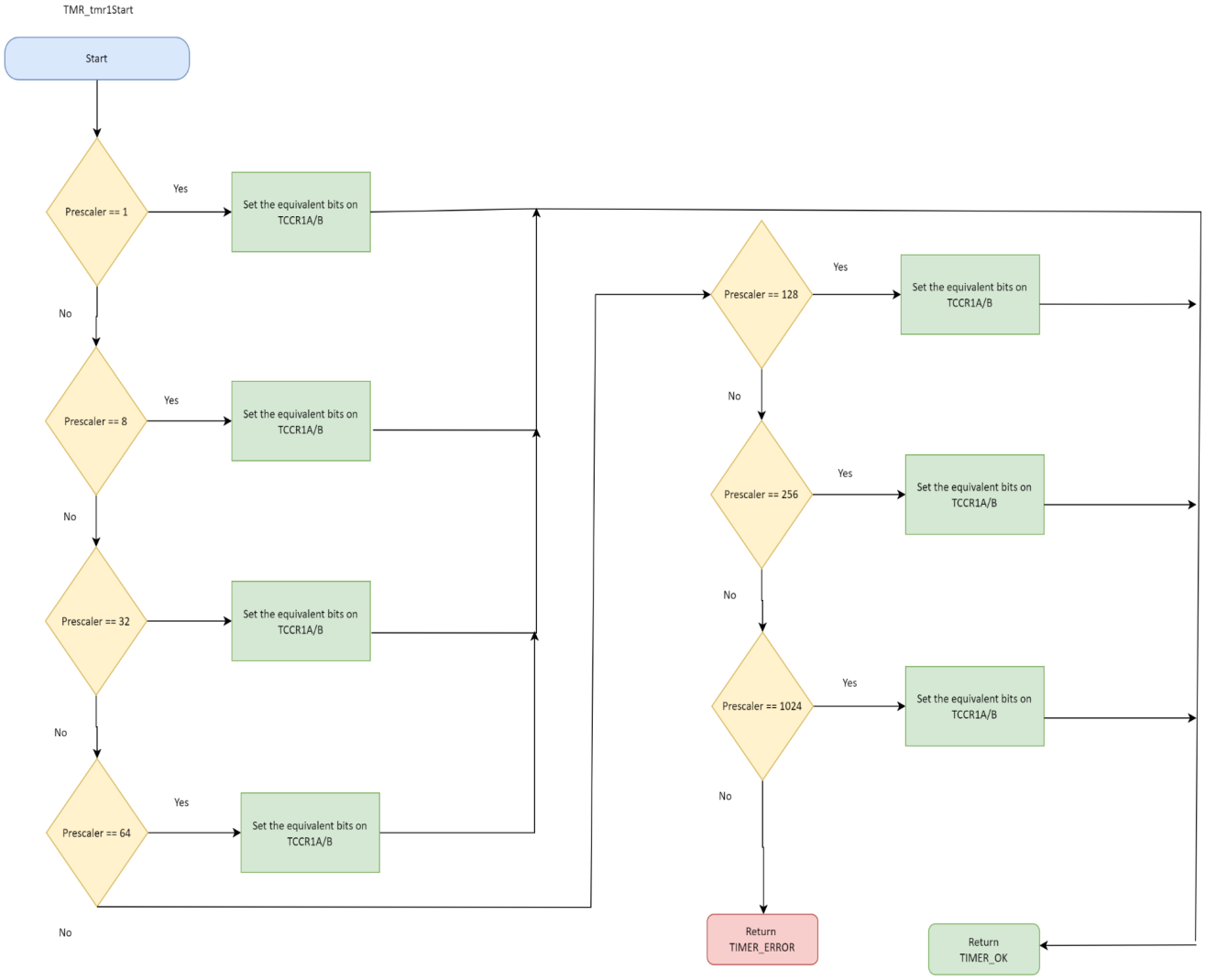
3.1.3.4. TIMER_TMR1NormaModeInit



3.1.3.4. Timer_TMR1Stop & ISR(TIM1_OVF_INT)

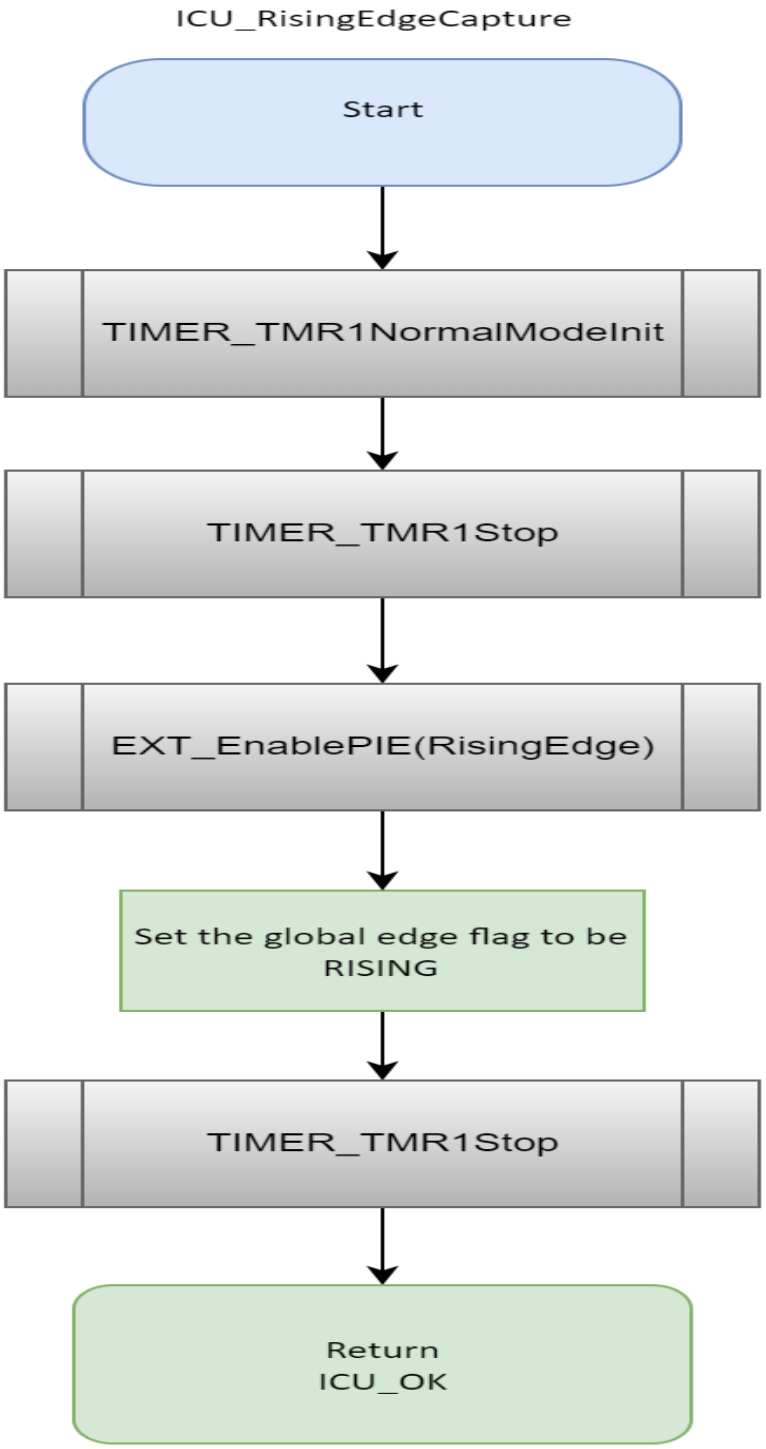


3.1.3.5 TIMER_TMR1Start

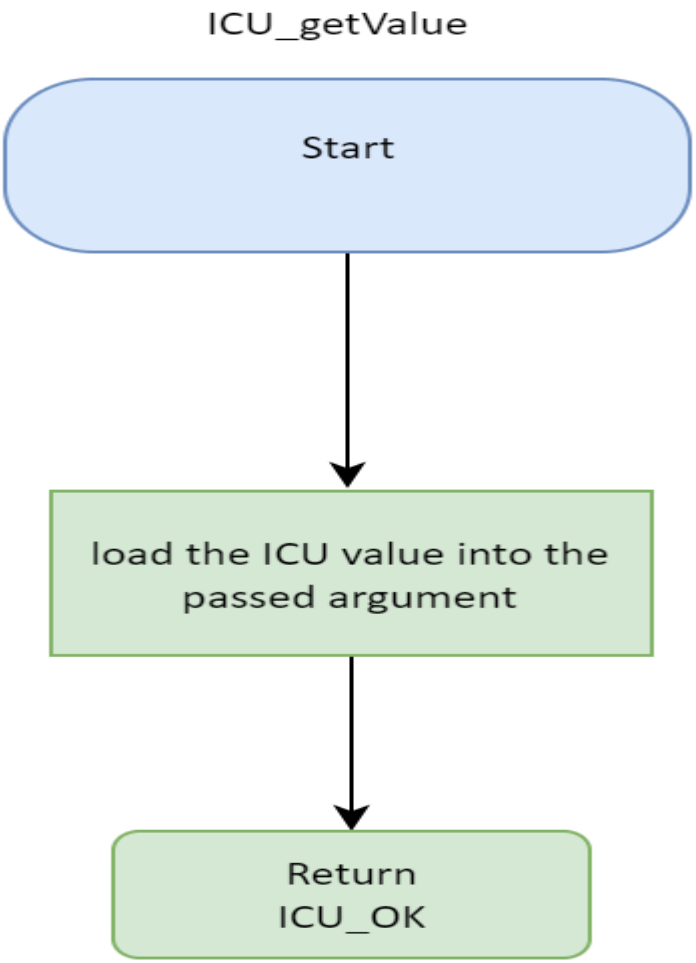


3.1.4. ICU

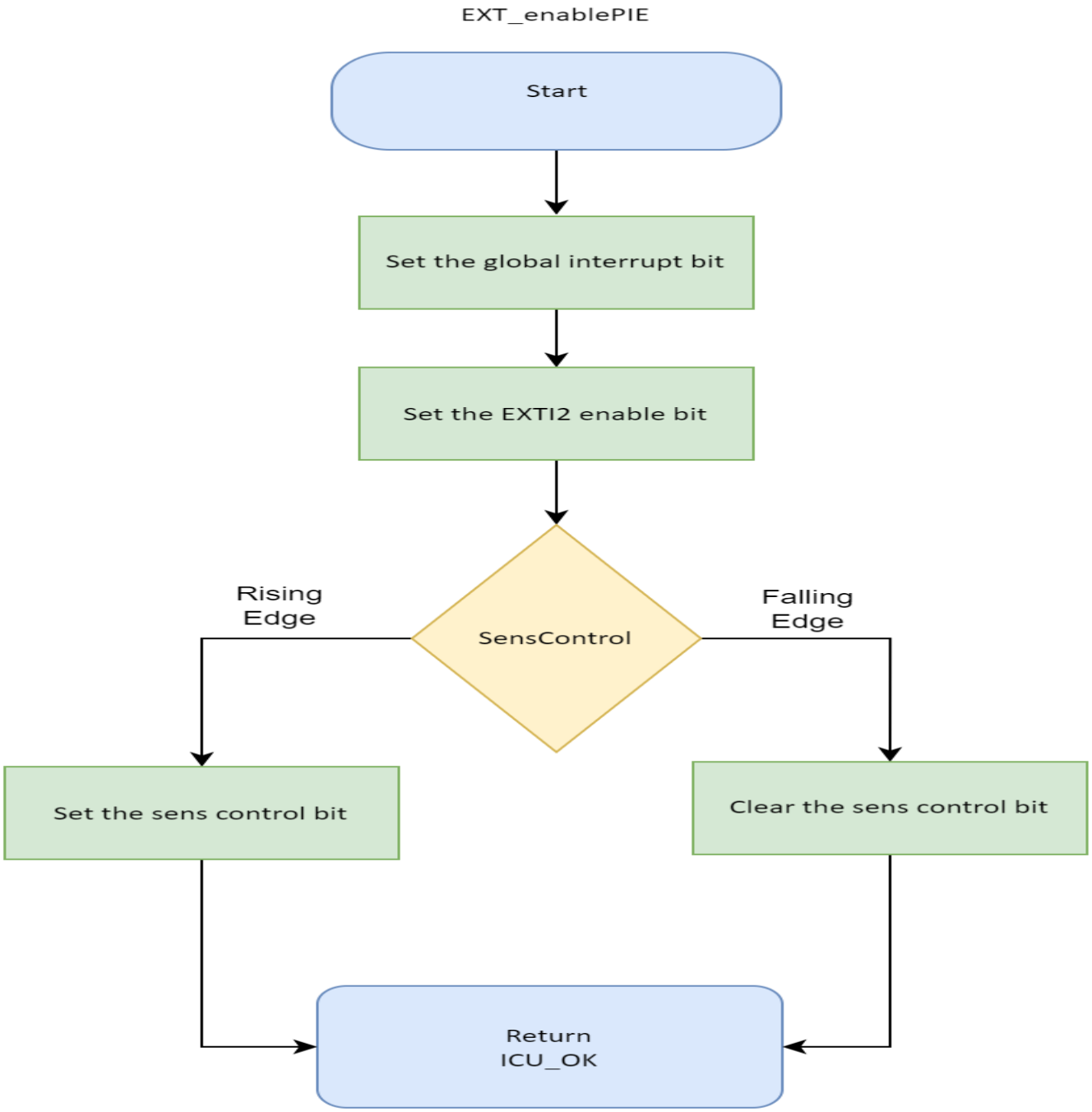
3.1.4.1 ICU_risingEgeCapture



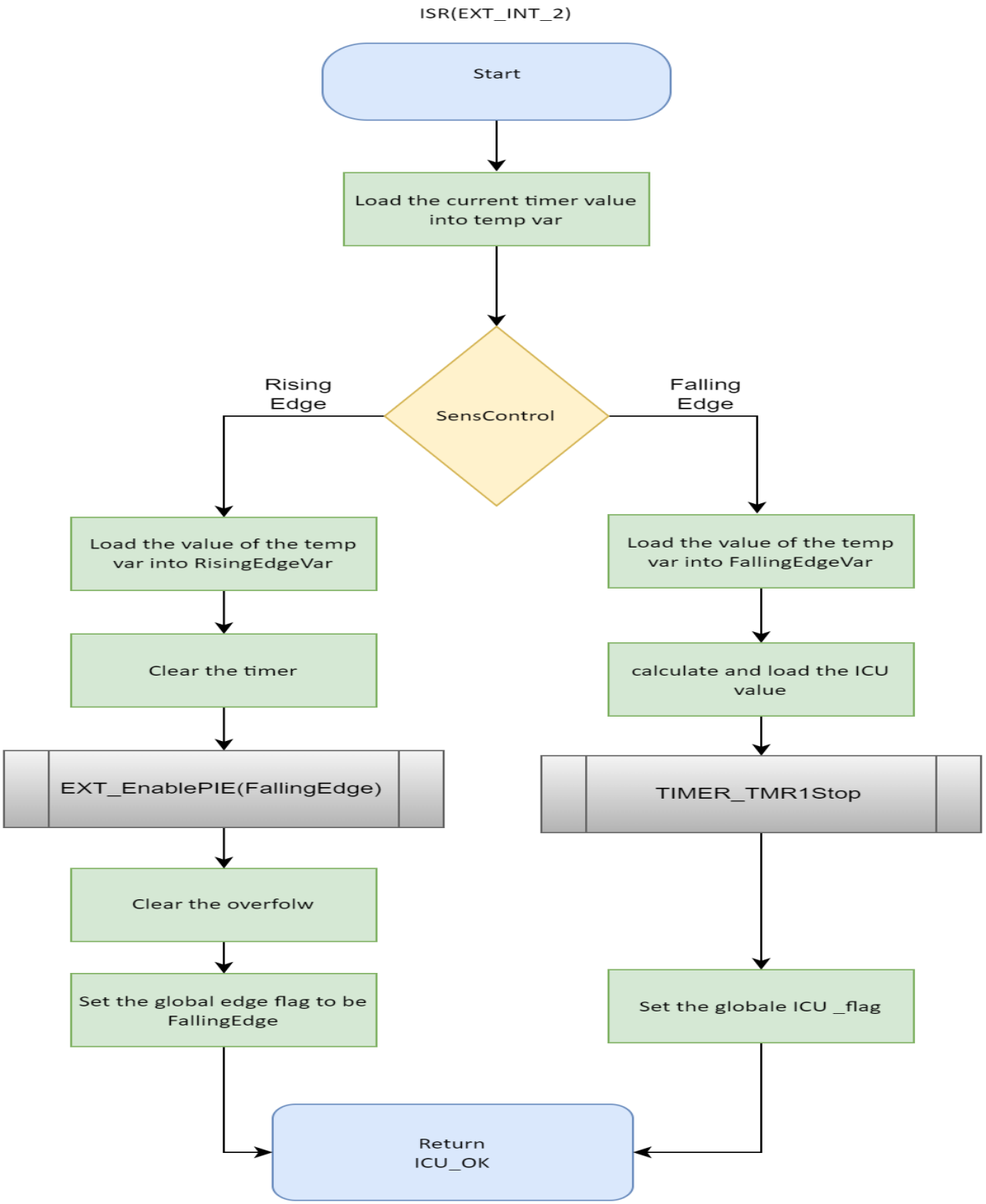
3.1.4.2. ICU_getValue



3.1.4.3. ICU_enablePIE

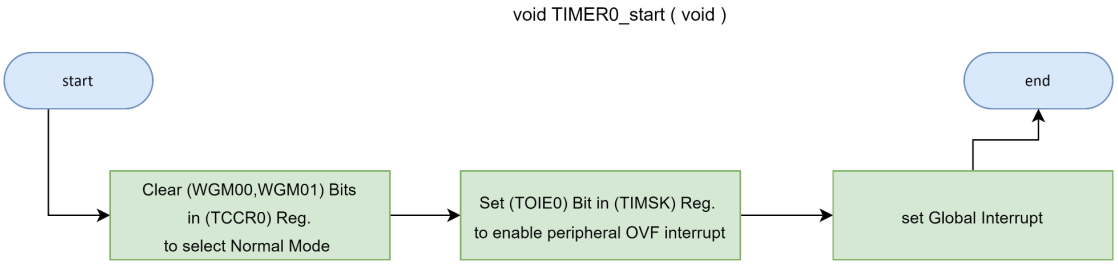


3.1.4.4. ISR(EXT_INT_2)

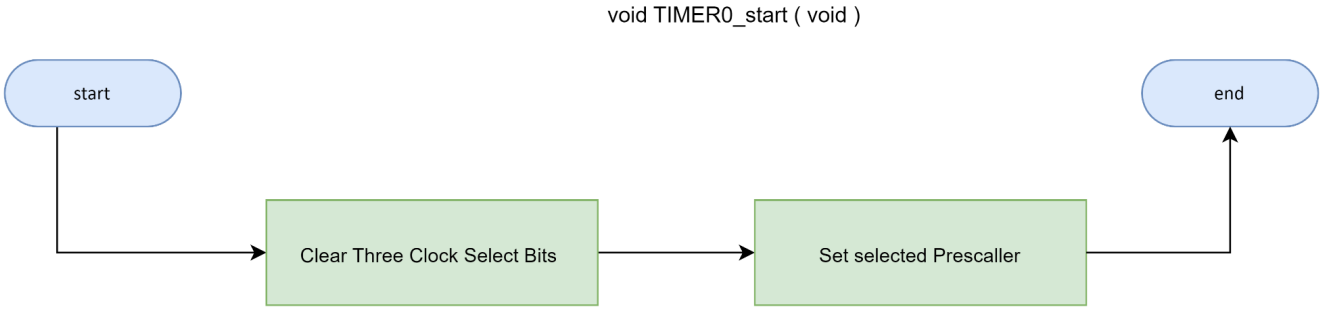


3.1.5. PWM

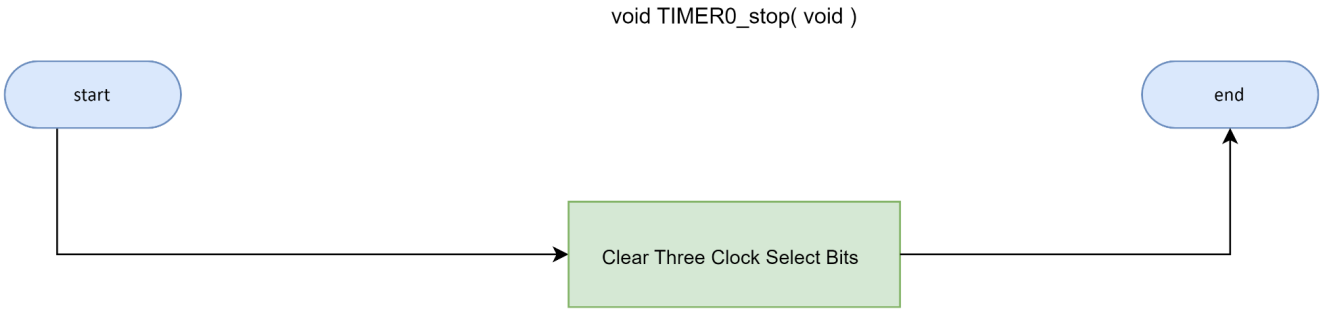
3.1.5.1 TIMER0_init



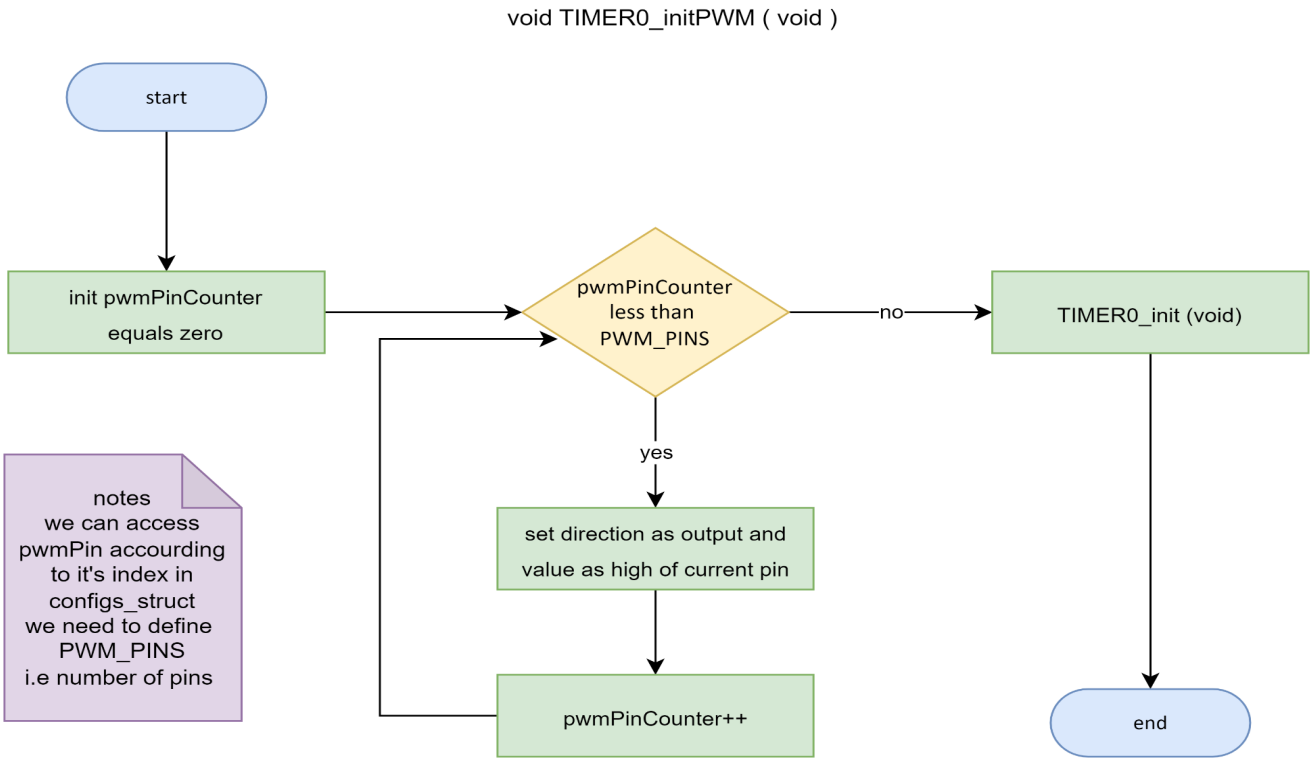
3.1.5.2 TIMER0_start



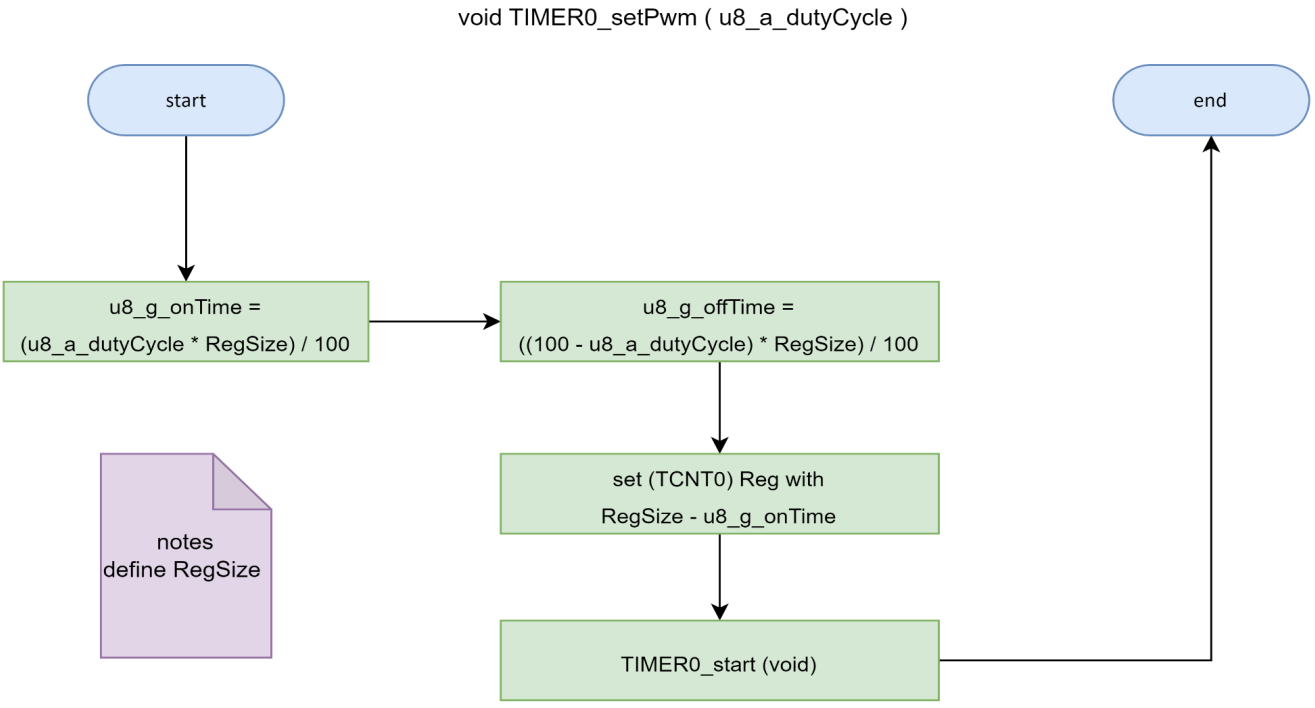
3.1.5.3 TIMER0_stop



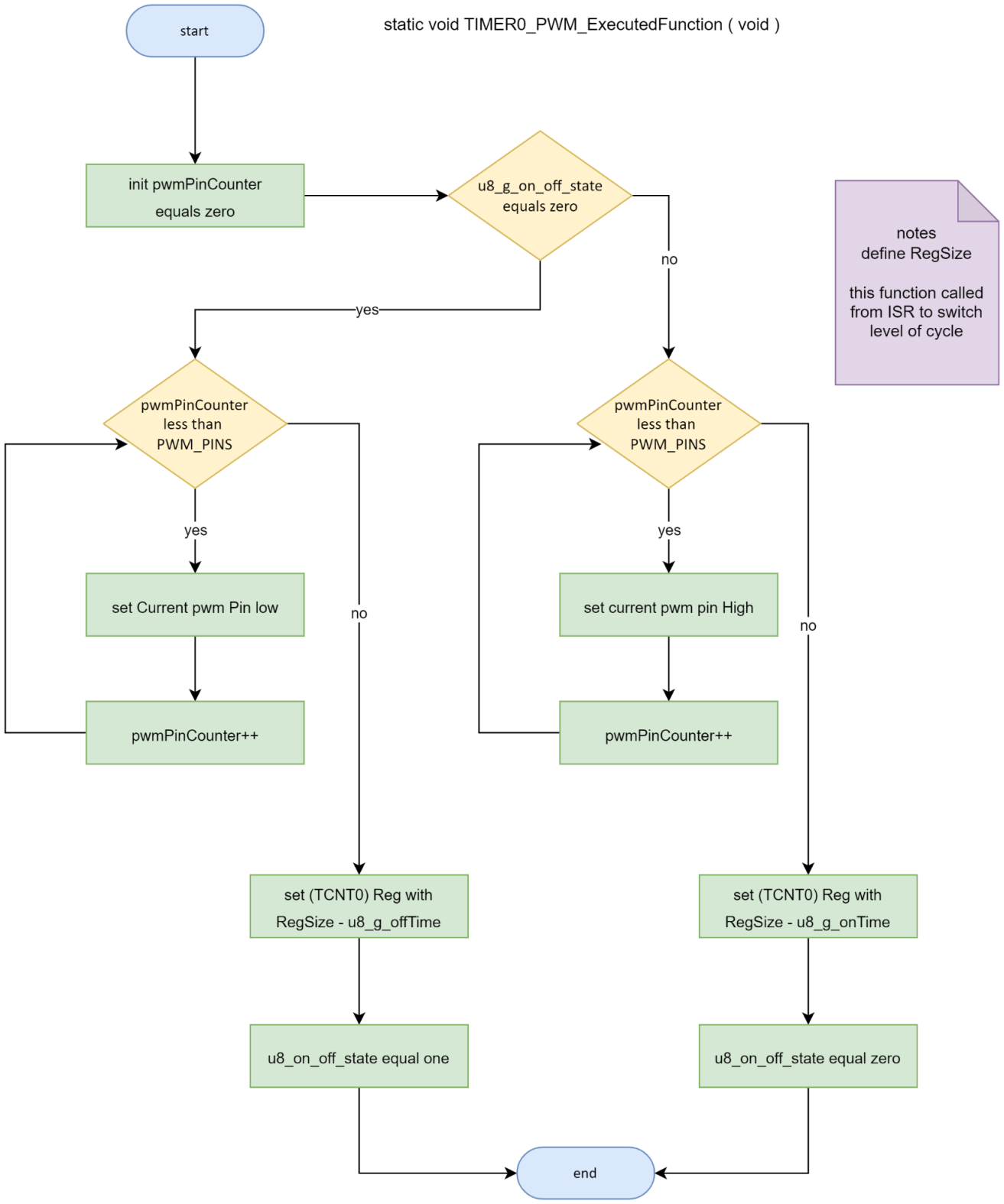
3.1.5.4 TIMER0_initPWM



3.1.5.5 TIMER0_setPwm



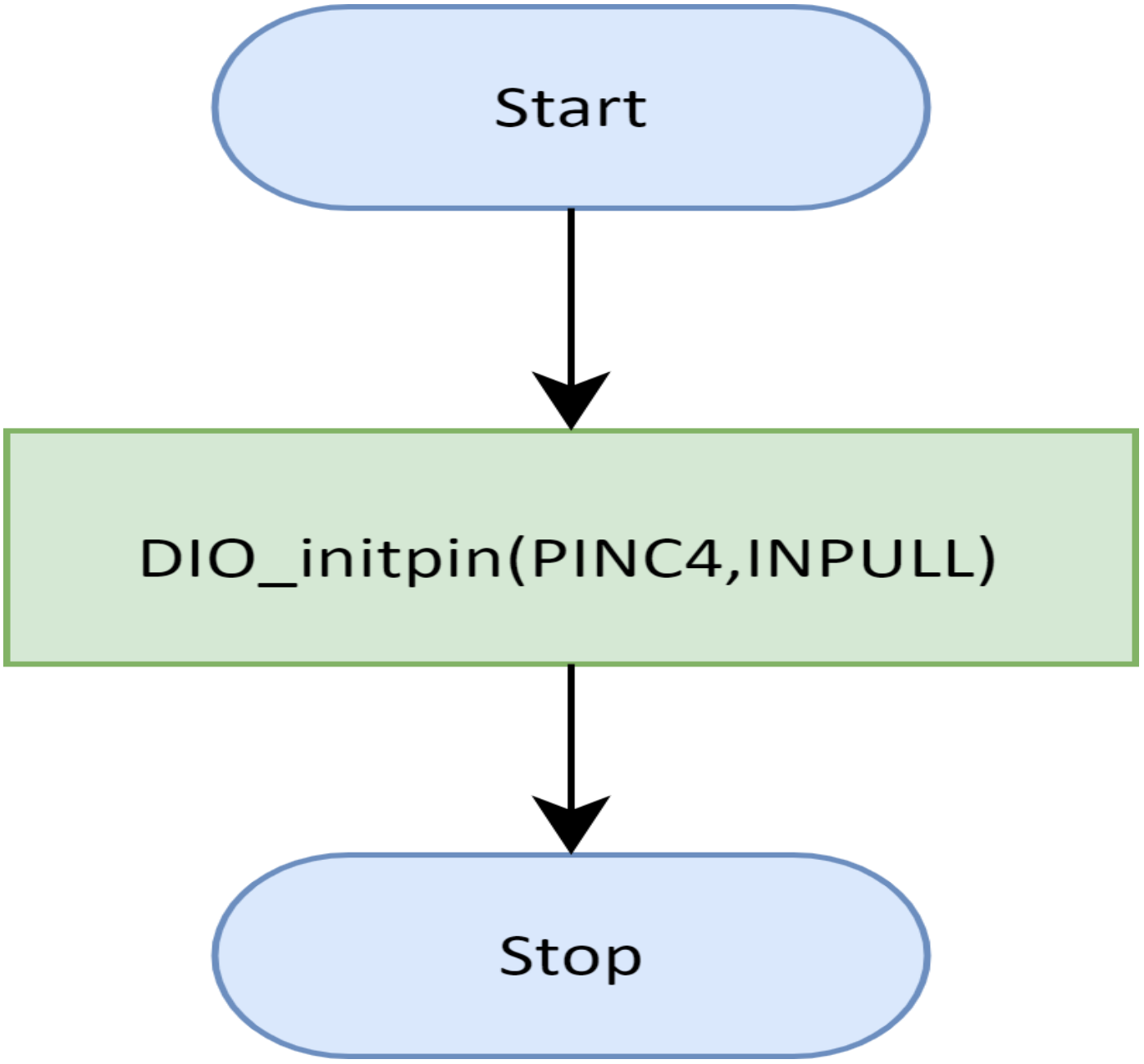
3.1.5.6 TIMER0_PWM_ExecutedFunction



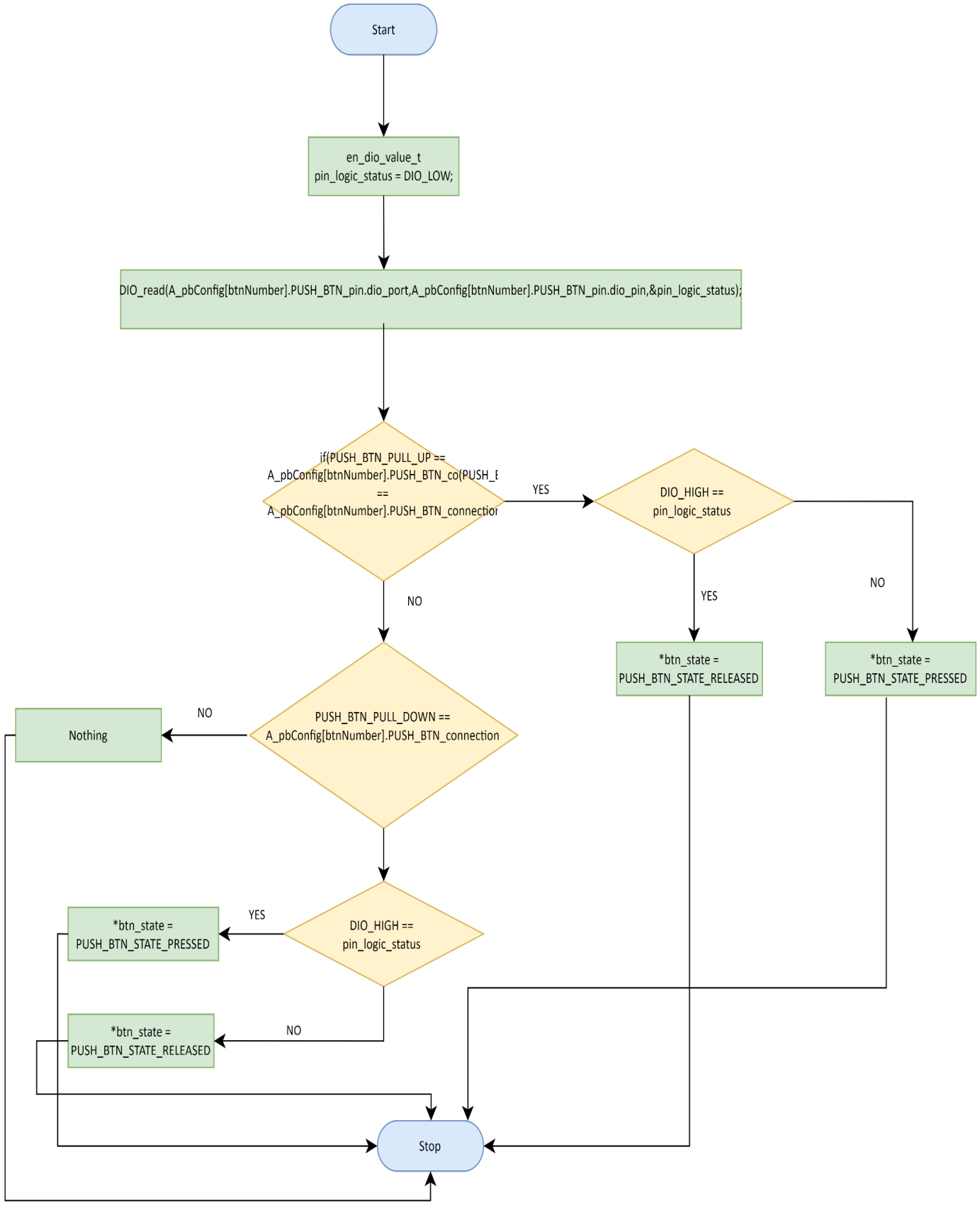
3.2. ECUAL

3.3.1. Push-BUTTON

3.3.1.1 PUSH_BTN_intialize

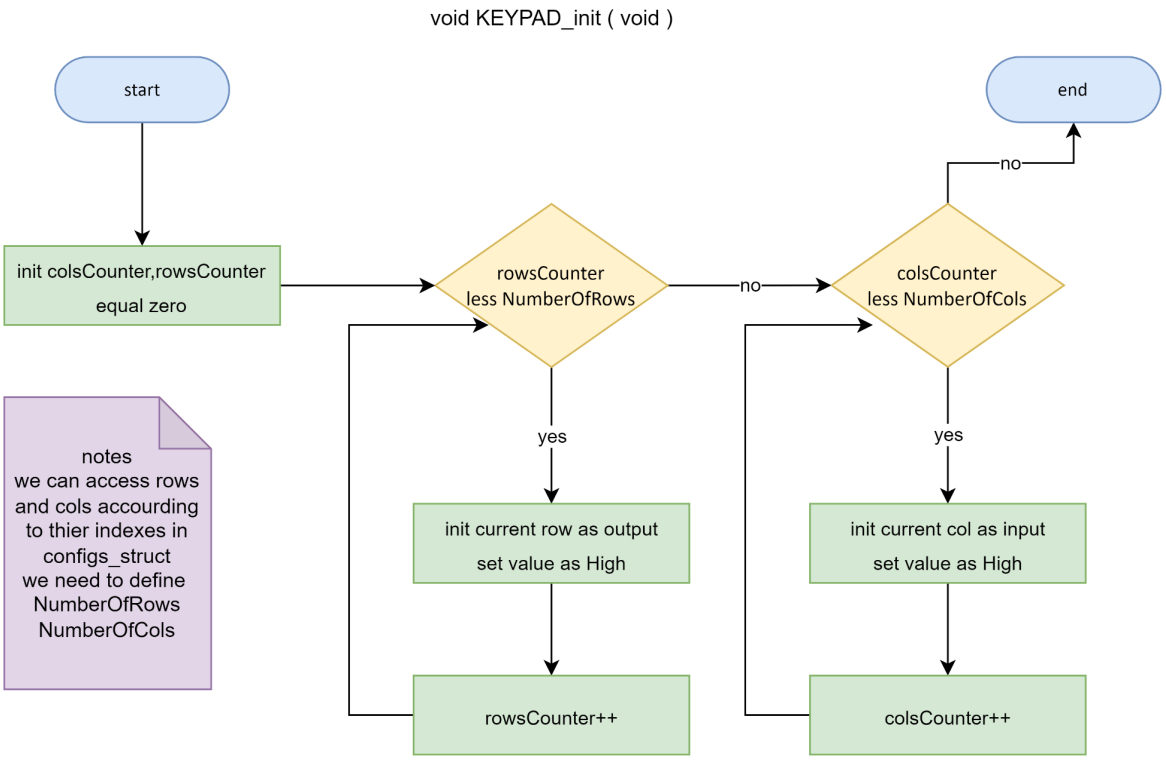


3.3.1.2 PUSH_BTN_read_state

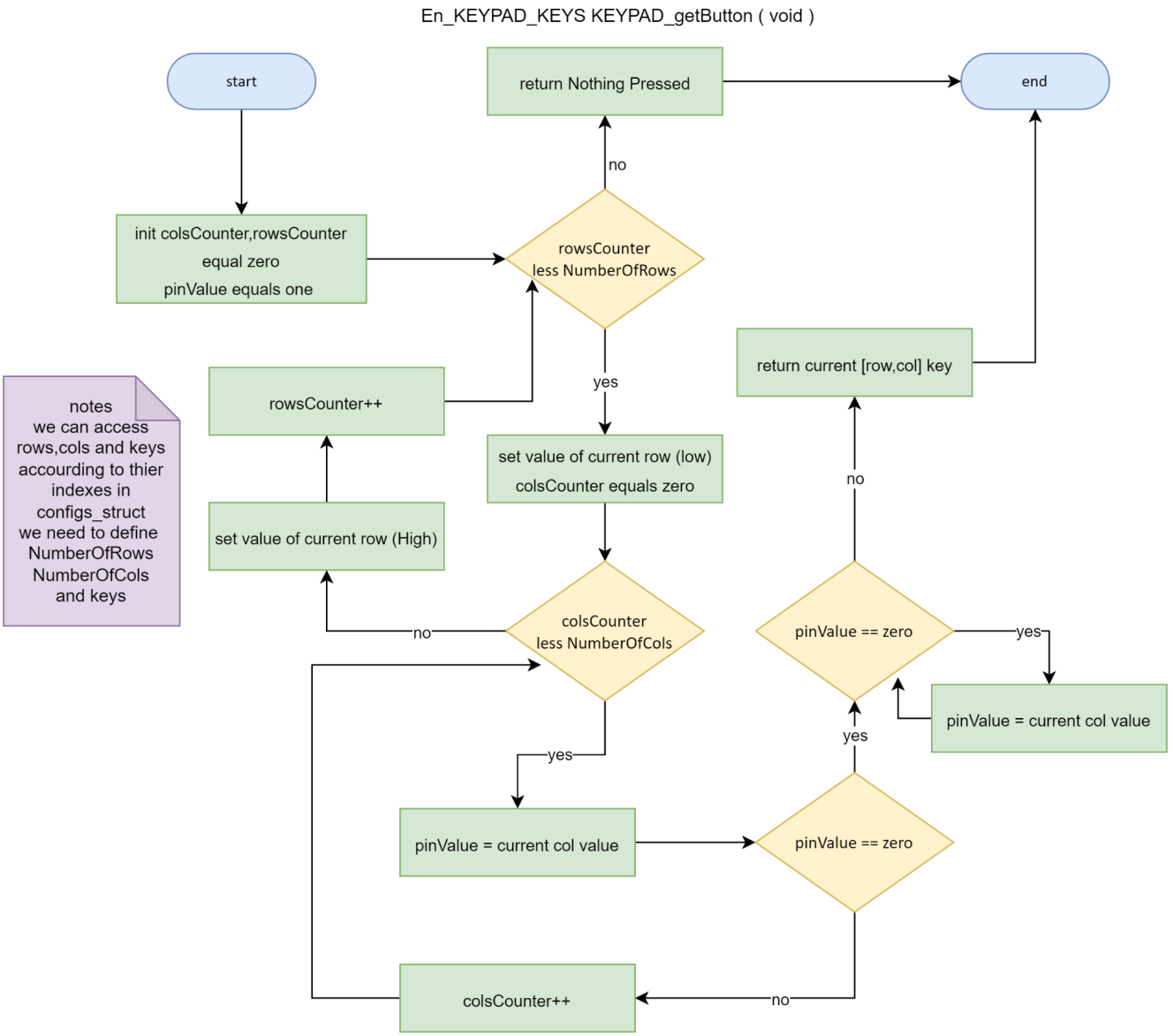


3.3.2. Keypad

3.3.2.1 KEYPAD_init

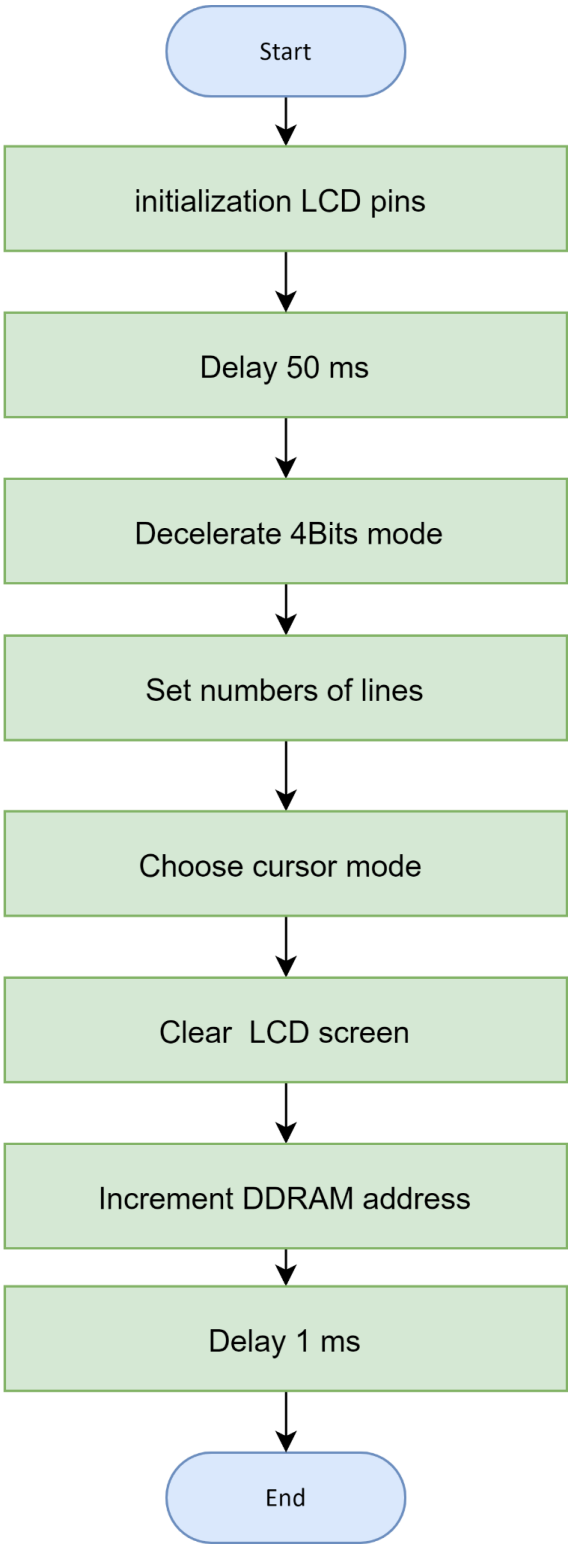


3.3.2.2 KEYPAD_getButton

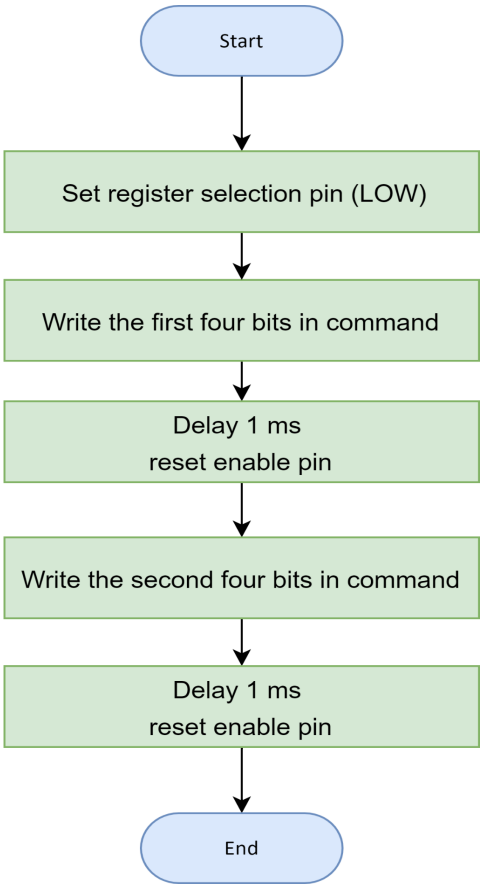

En_KEYPAD_KEYS KEYPAD_getButton (void)
start
return Nothing Pressed
end
no
init colsCounter,rowsCounter
equal zero
pinValue equals one
rowsCounter
less NumberOfRows
yes
return current [row,col] key
notes
we can access
rows,cols and keys
accourding to thier
indexes in
configs_struct
we need to define
NumberOfRows
NumberOfCols
and keys
rowsCounter++
set value of current row (low)
colsCounter equals zero
no
set value of current row (High)
colsCounter
less NumberOfCols
pinValue == zero
yes
pinValue = current col value
yes
yes
pinValue = current col value
pinValue == zero
colsCounter++
no

3.3.3. LCD

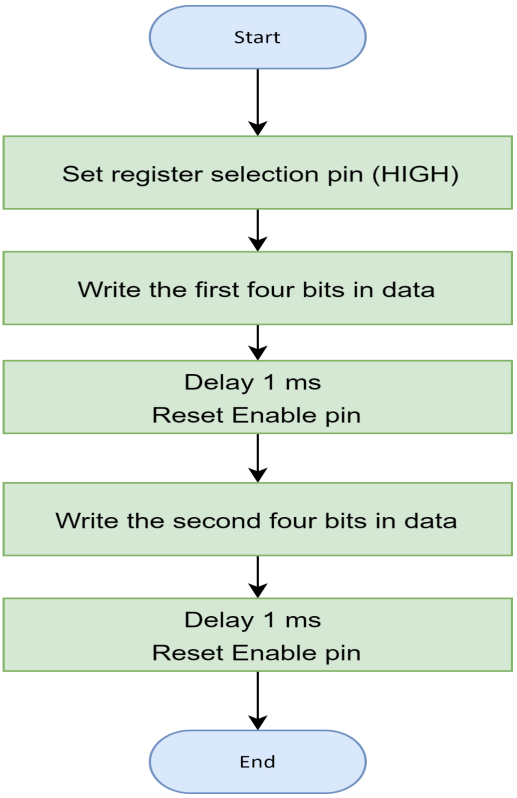
3.3.3.1.LCD_Init



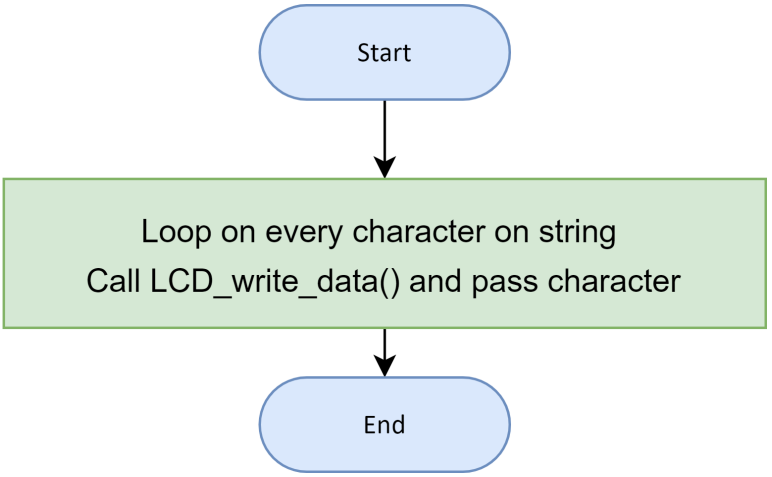
3.3.3.2.LCD_write_ins



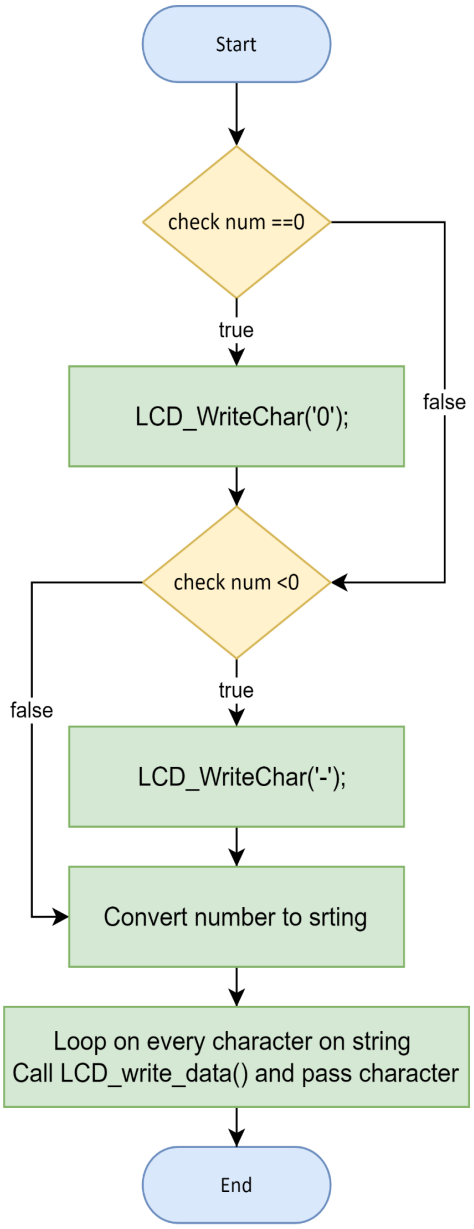
3.3.3.3.LCD_write_data



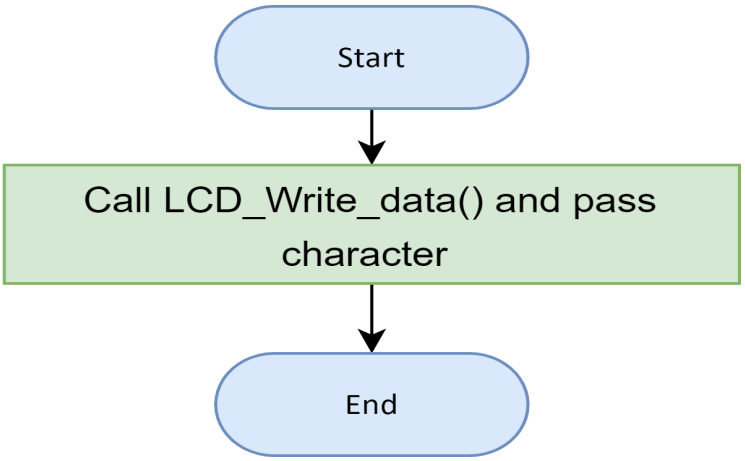
3.3.3.4.LCD_write_string



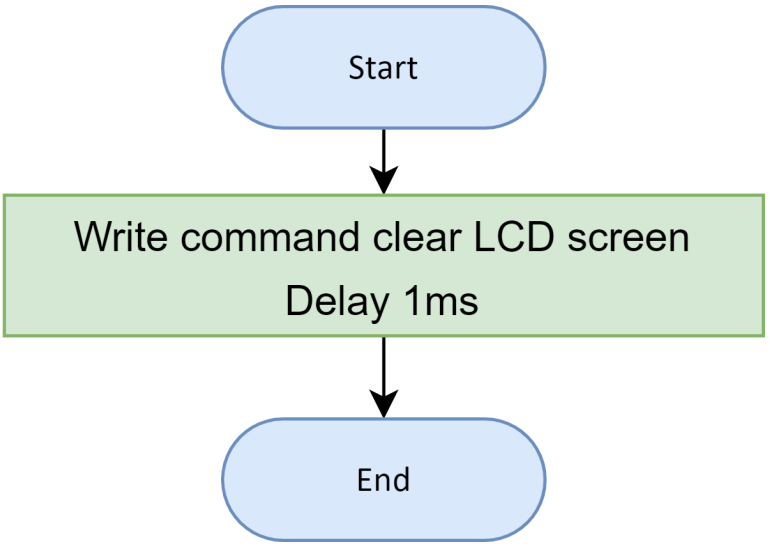
3.3.3.5.LCD_write_number



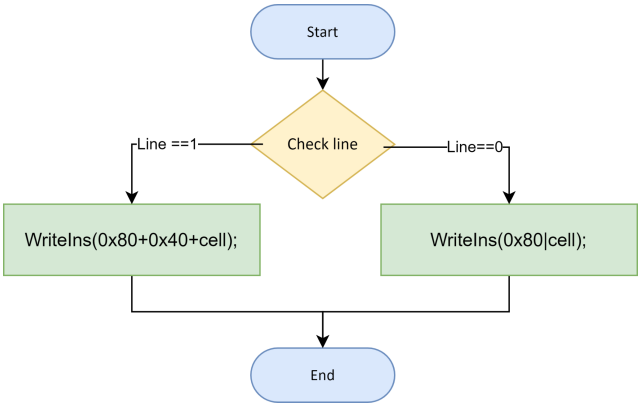
3.3.3.6.LCD_write_Char



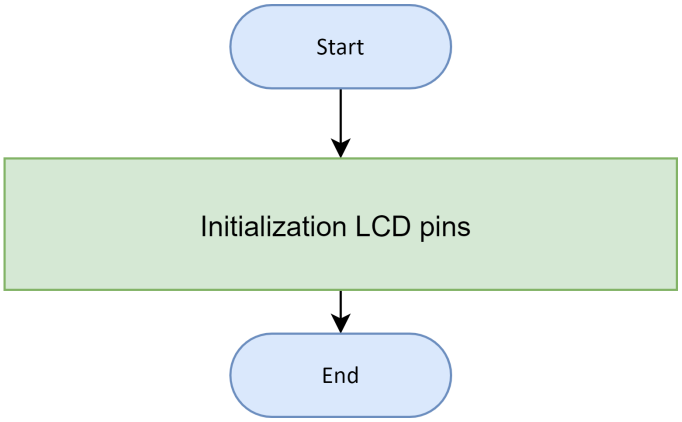
3.3.3.7.LCD_Clear



3.3.3.8.LCD_Setcursor

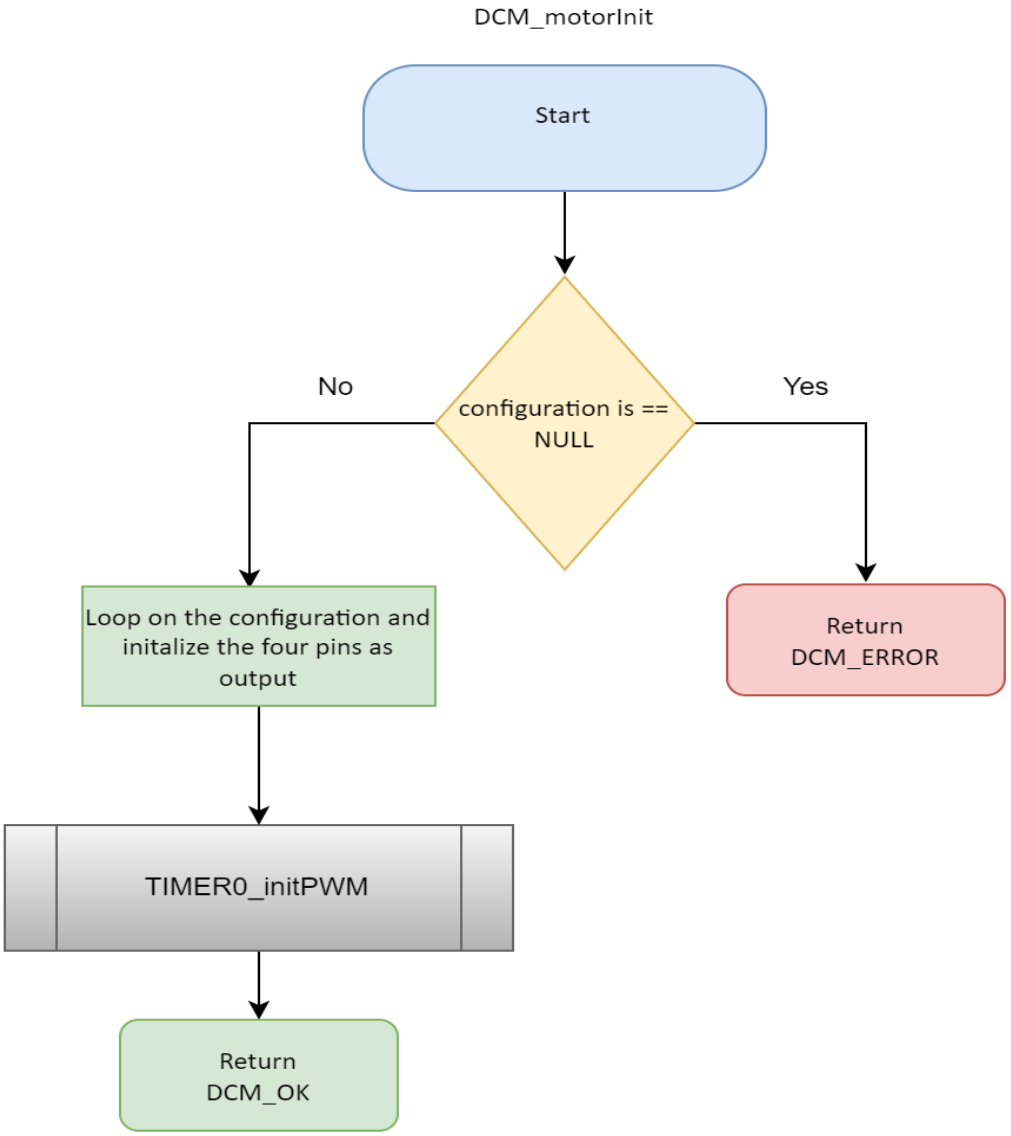


3.3.3.9.LCD_init_pins

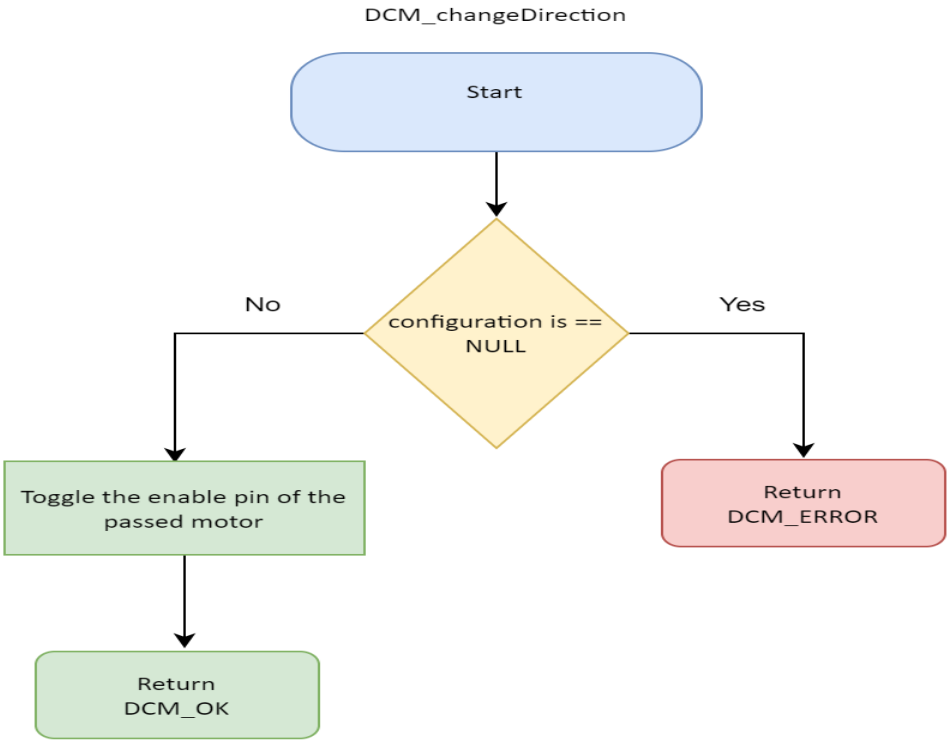


3.3.4. DCM

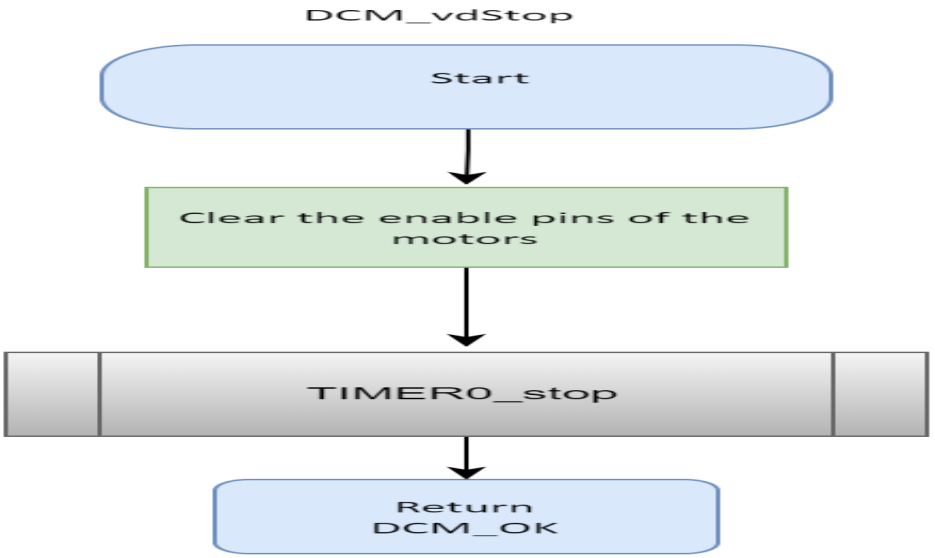
3.3.4.1.DCM_motorInit



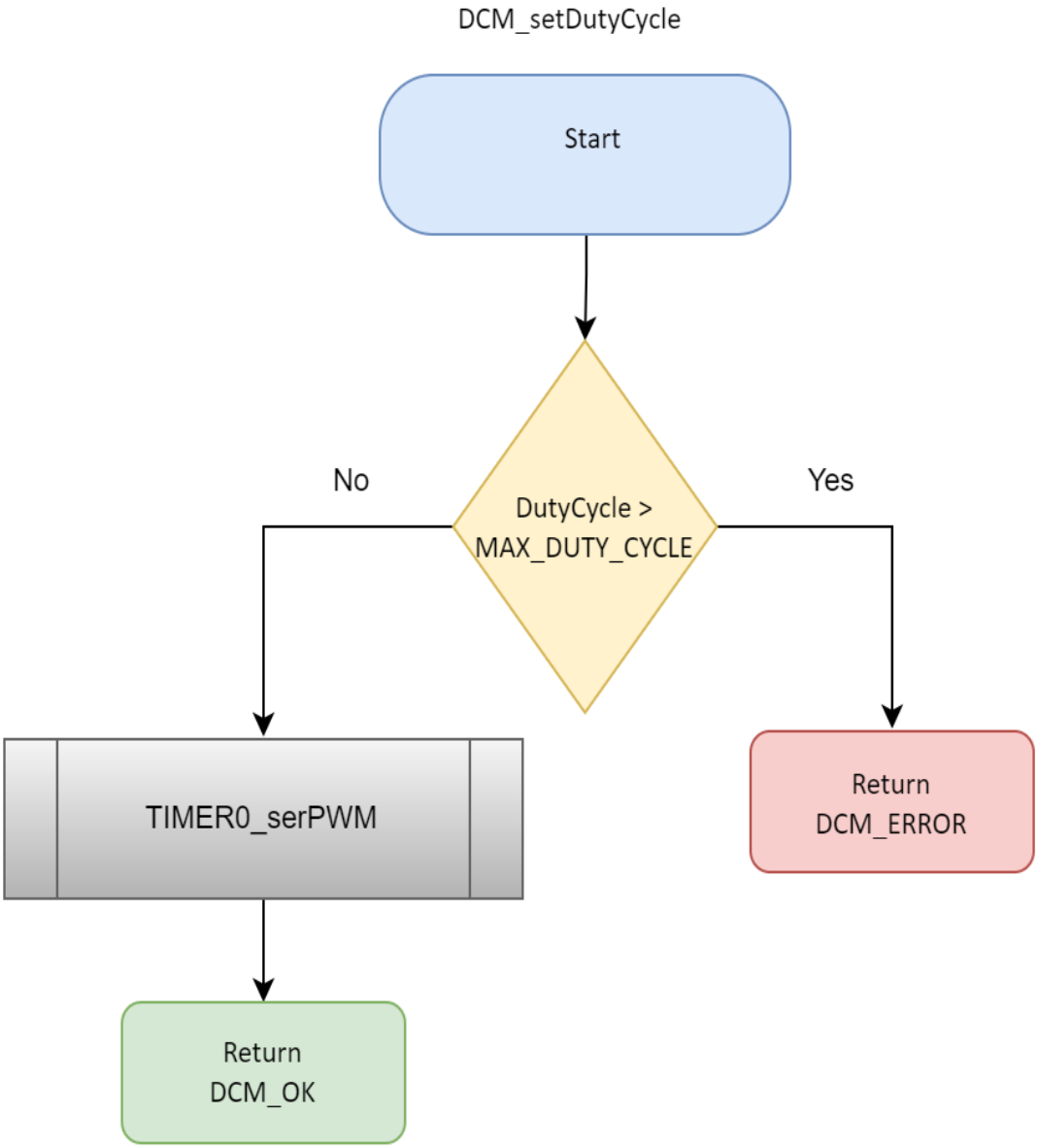
3.3.4.2.DCM_changeDirection



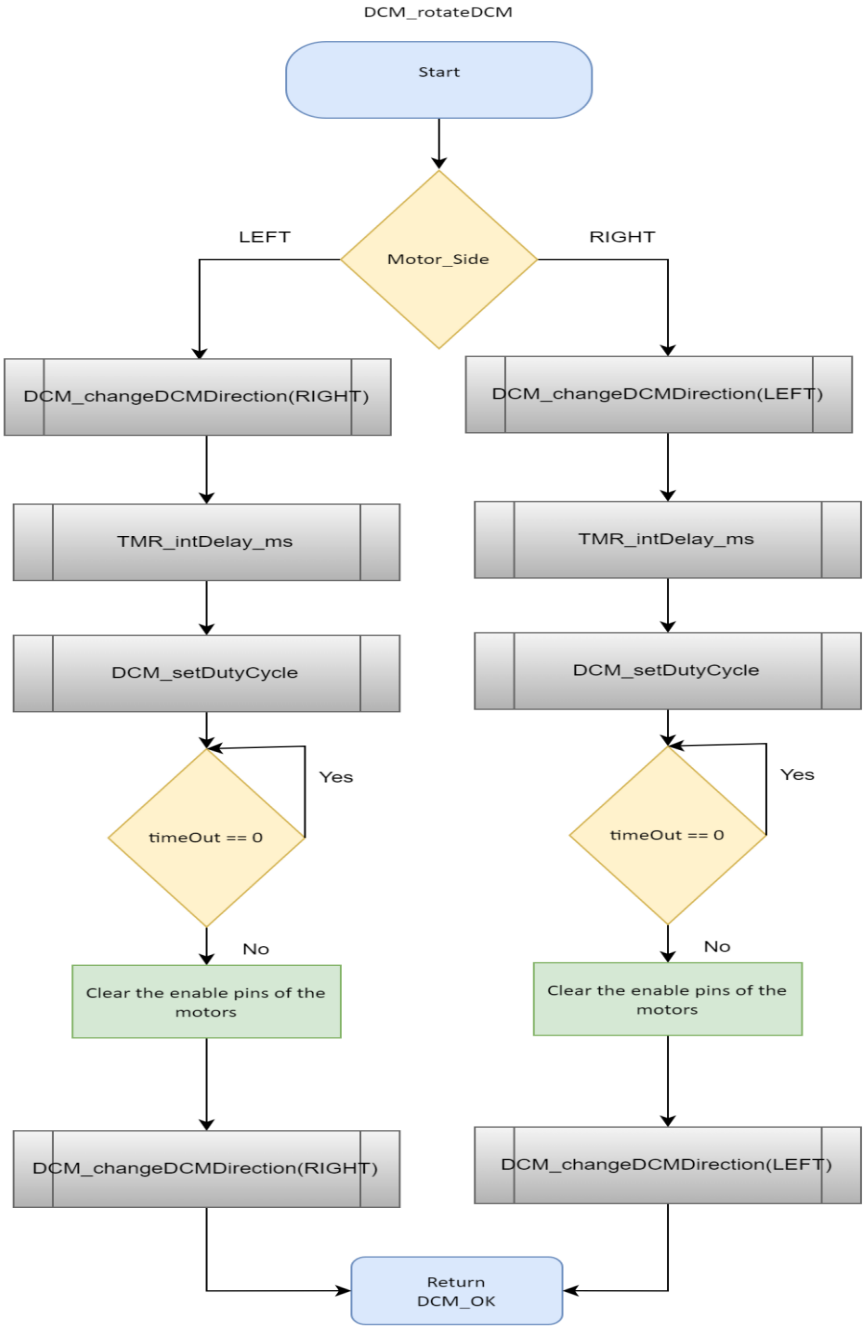
3.3.4.3.DCM_vdStop



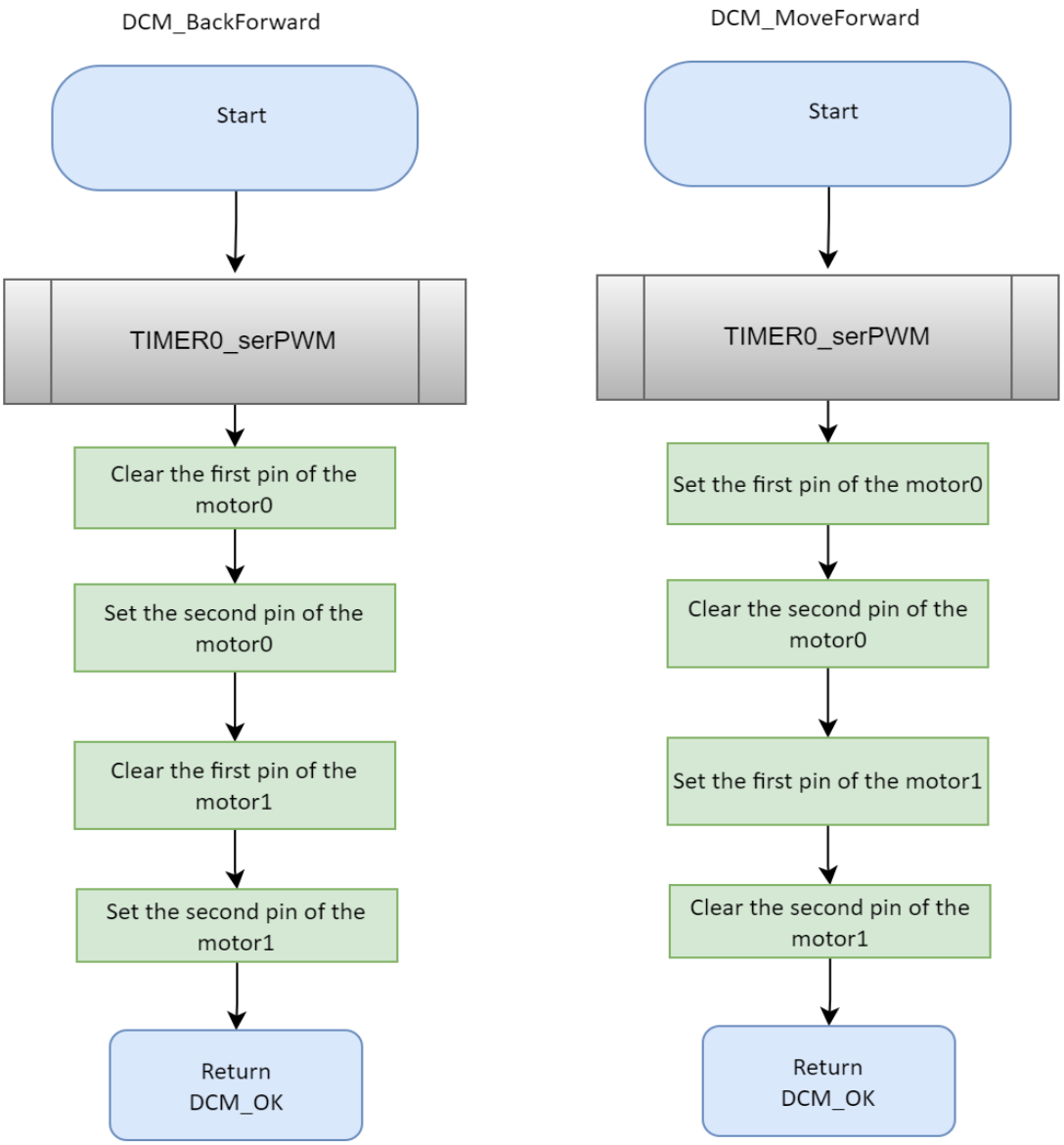
3.3.4.4.DCM_setDutyCycle



3.3.4.5.DCM_rotate

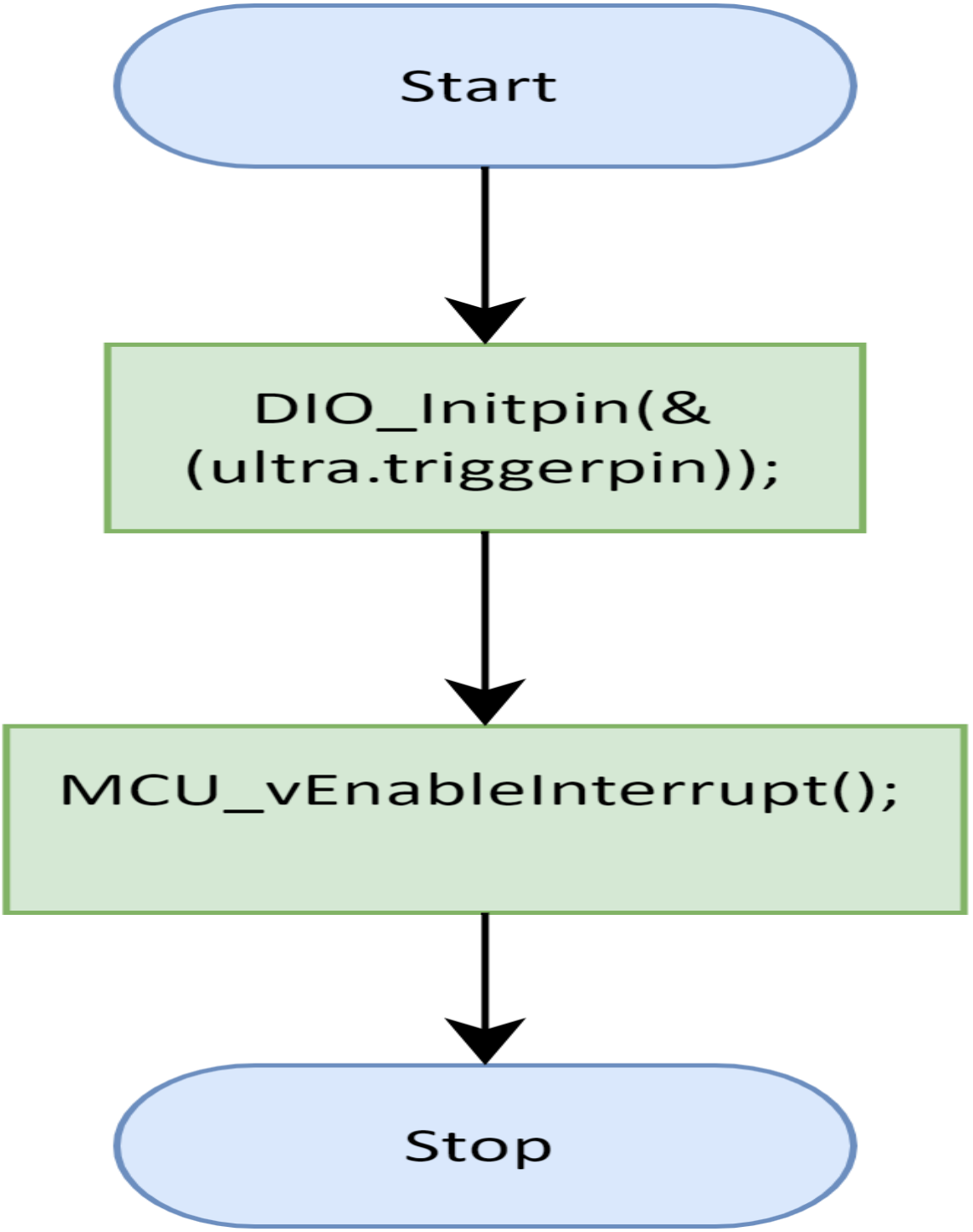


3.3.4.6. DCM_MoveForward / DCM_MoveBackward

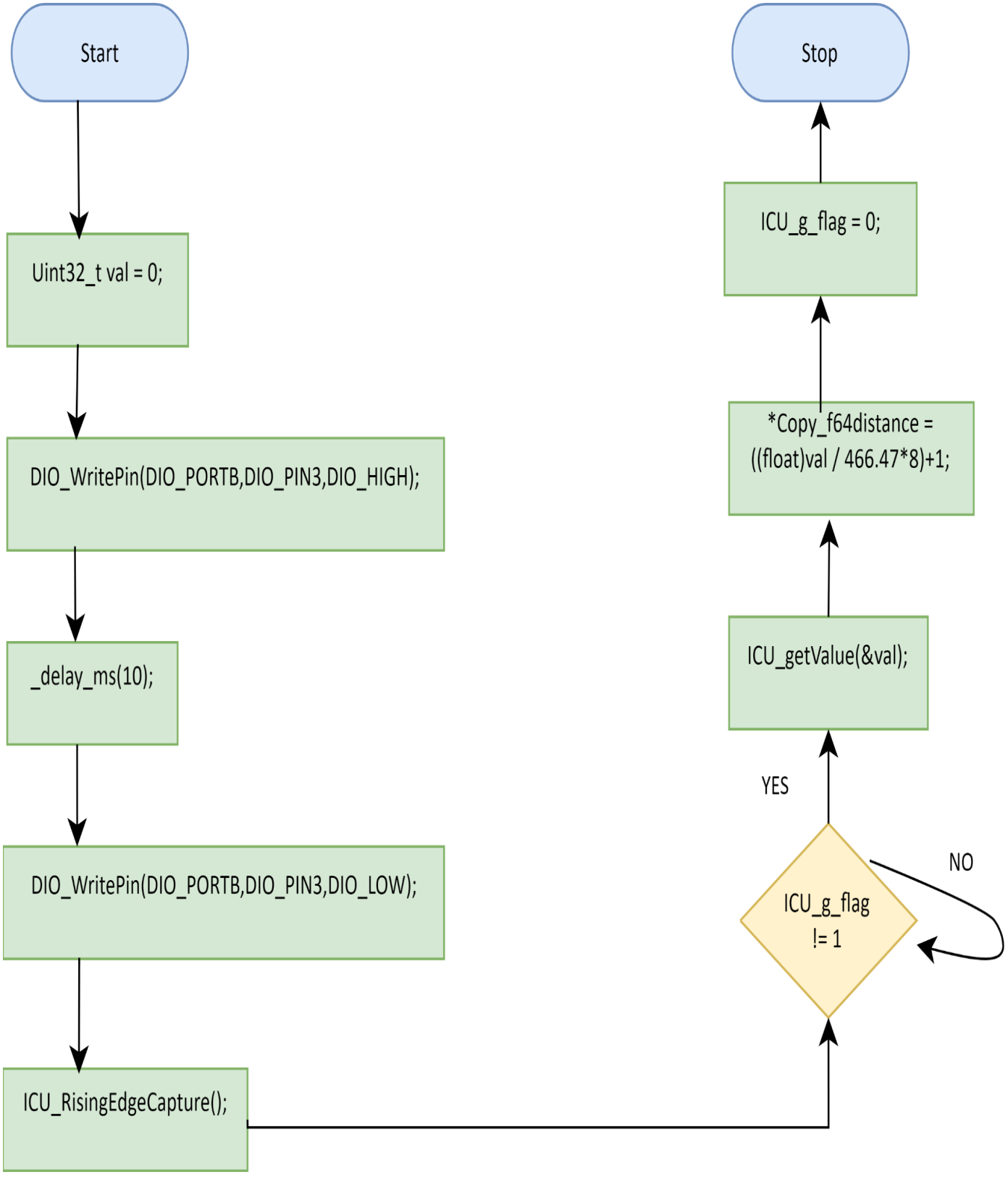


3.3.5 Ultrasonic

3.3.5.1 ultrasonic_vInit



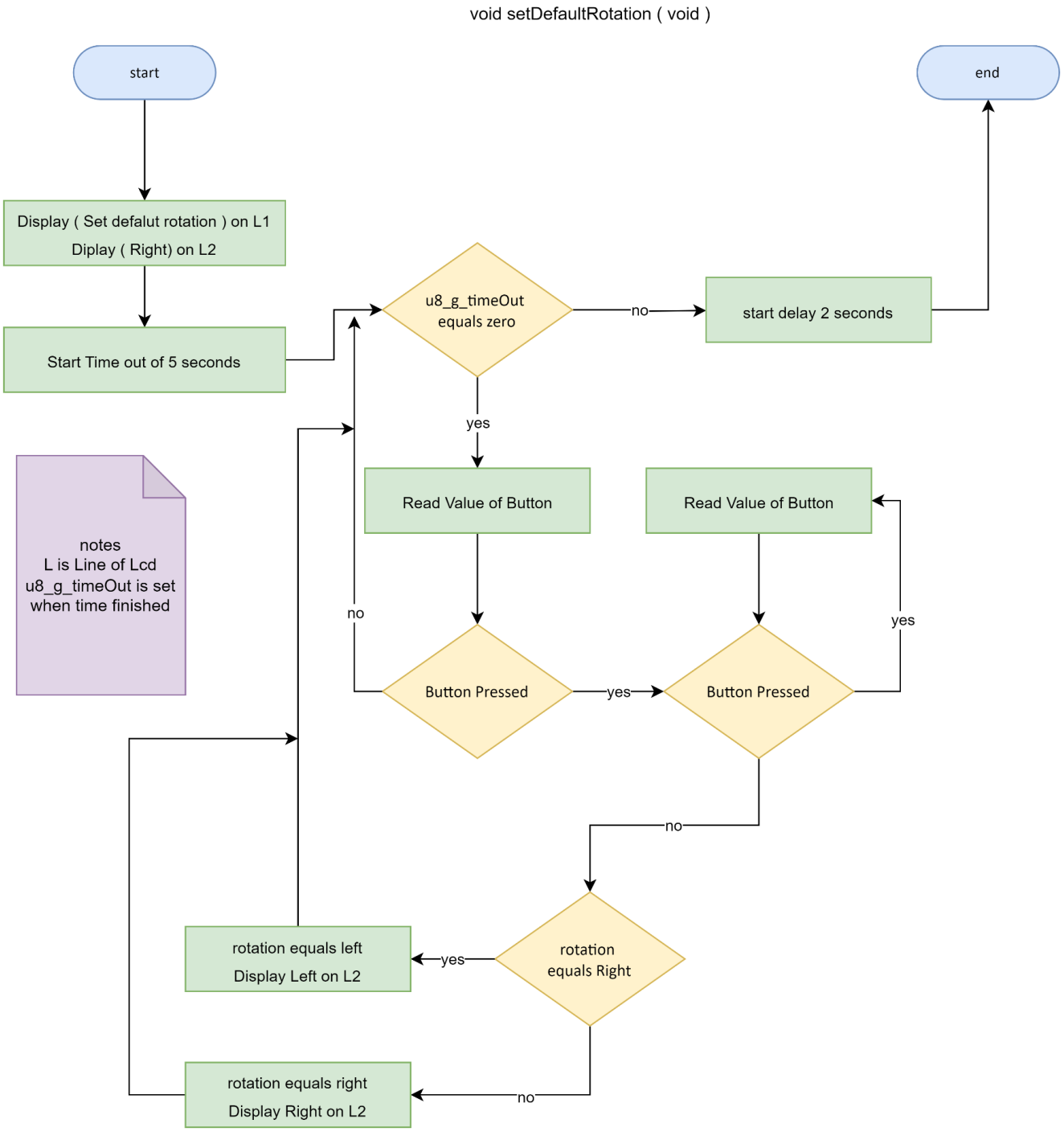
3.3.5.2 ultrasonic_vGetDistance



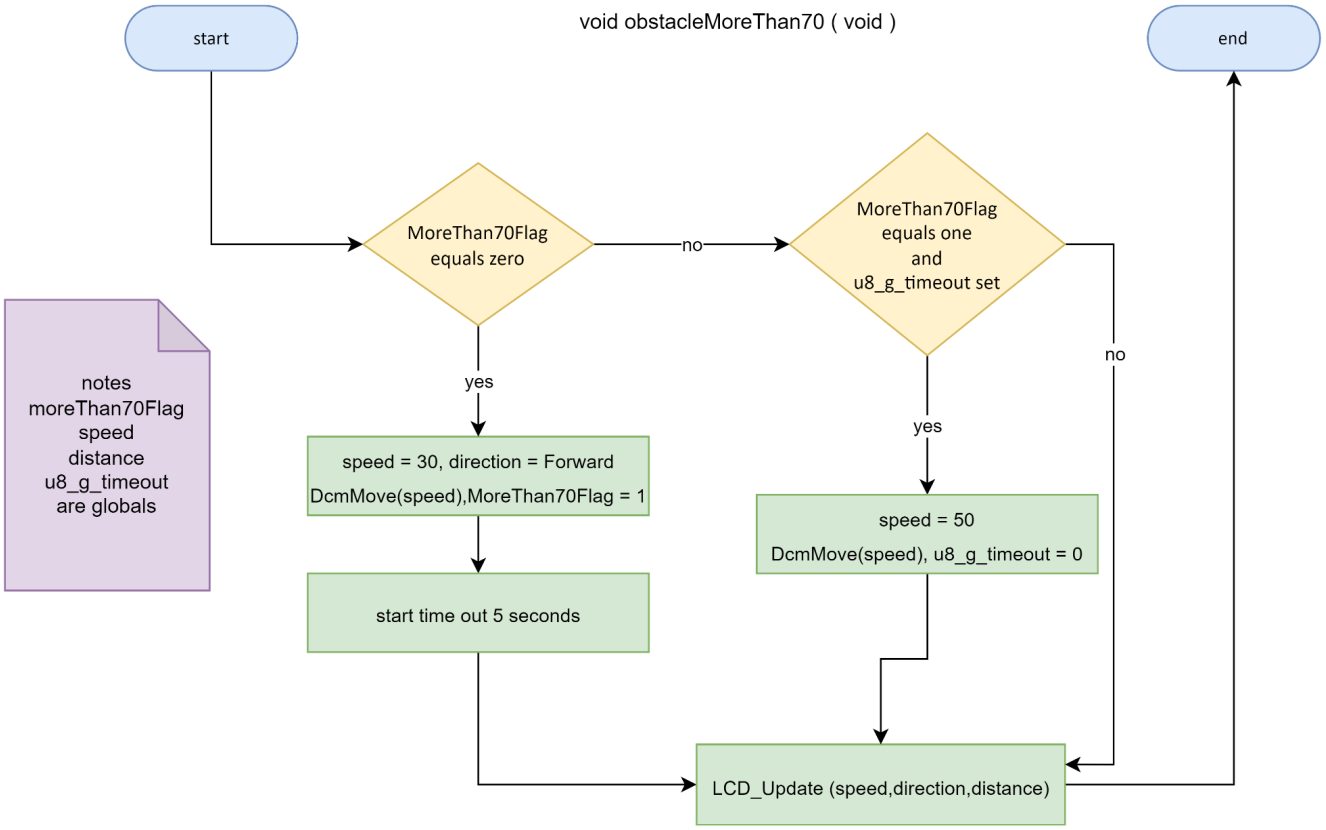
3.3. App

3.3.1. Car_operations

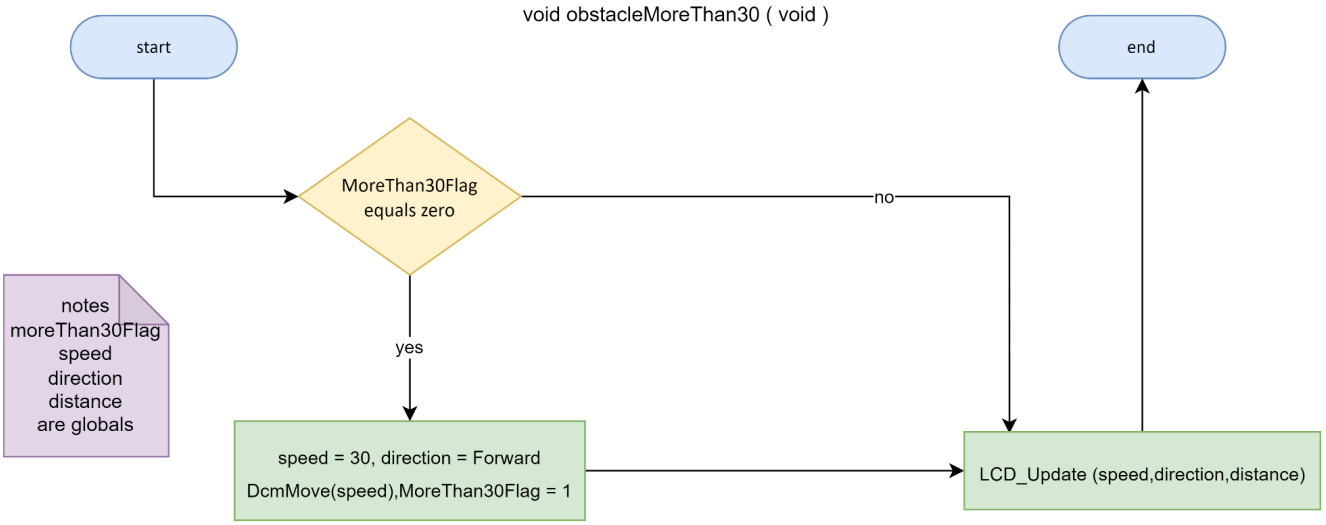
3.3.1.1.SetDefaultRotation



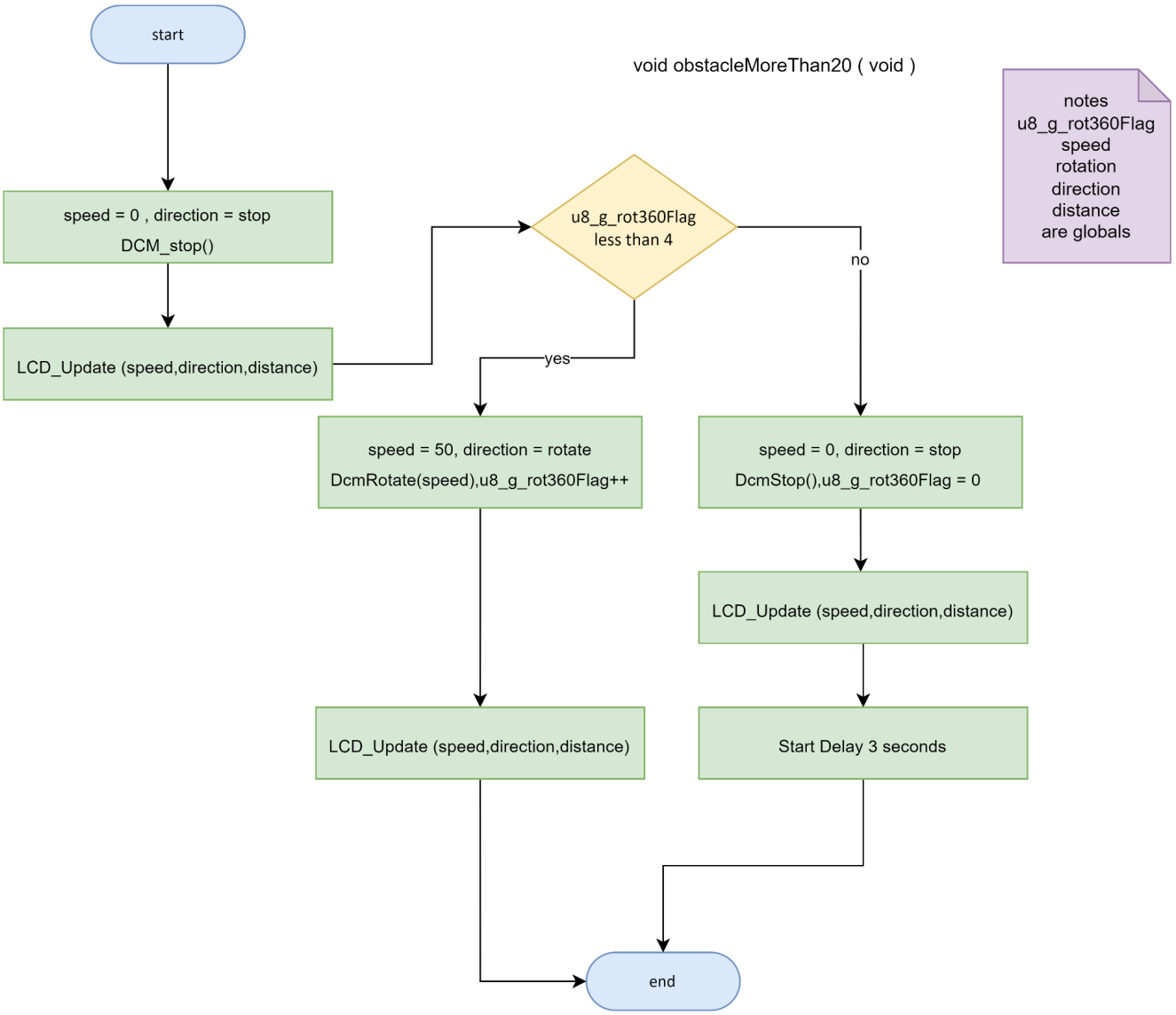
3.3.1.2.ObstacleMoreThan70



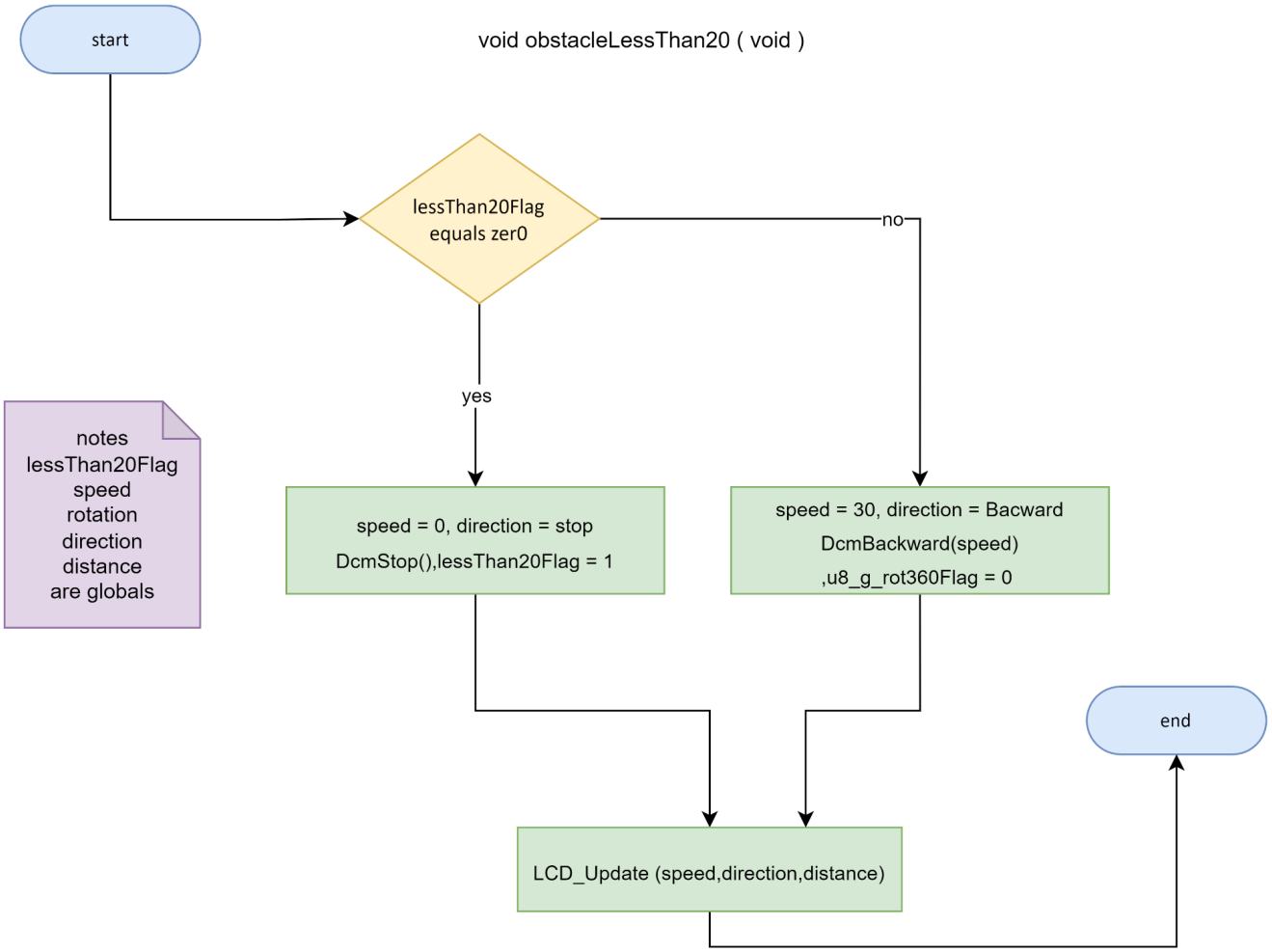
3.3.1.3.ObstacleMoreThan30



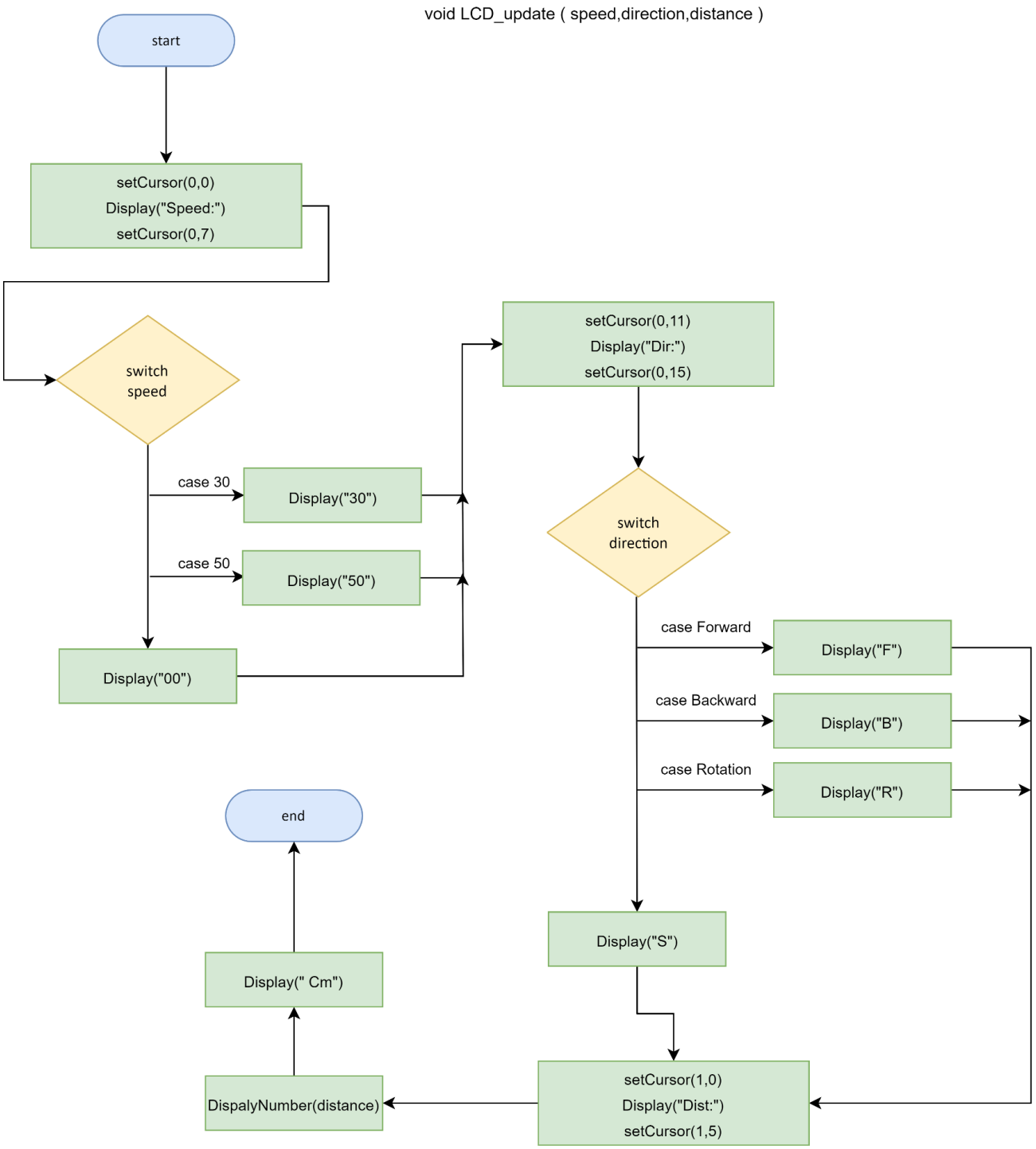
3.3.1.4.ObstacleMoreThan20



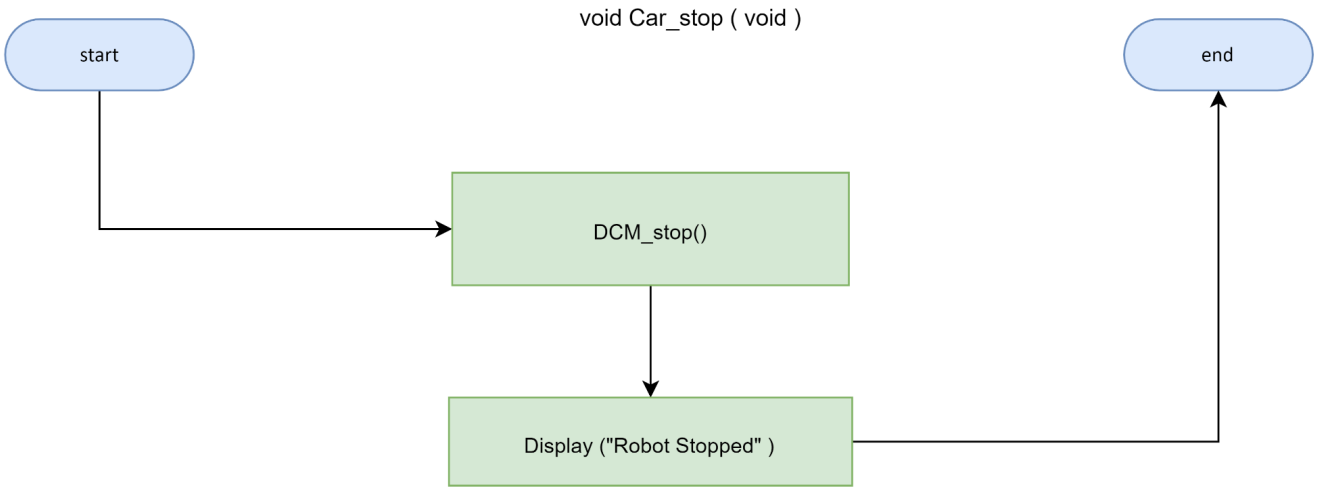
3.3.1.5.ObstacleLessThan20



3.3.1.6.LCD_update

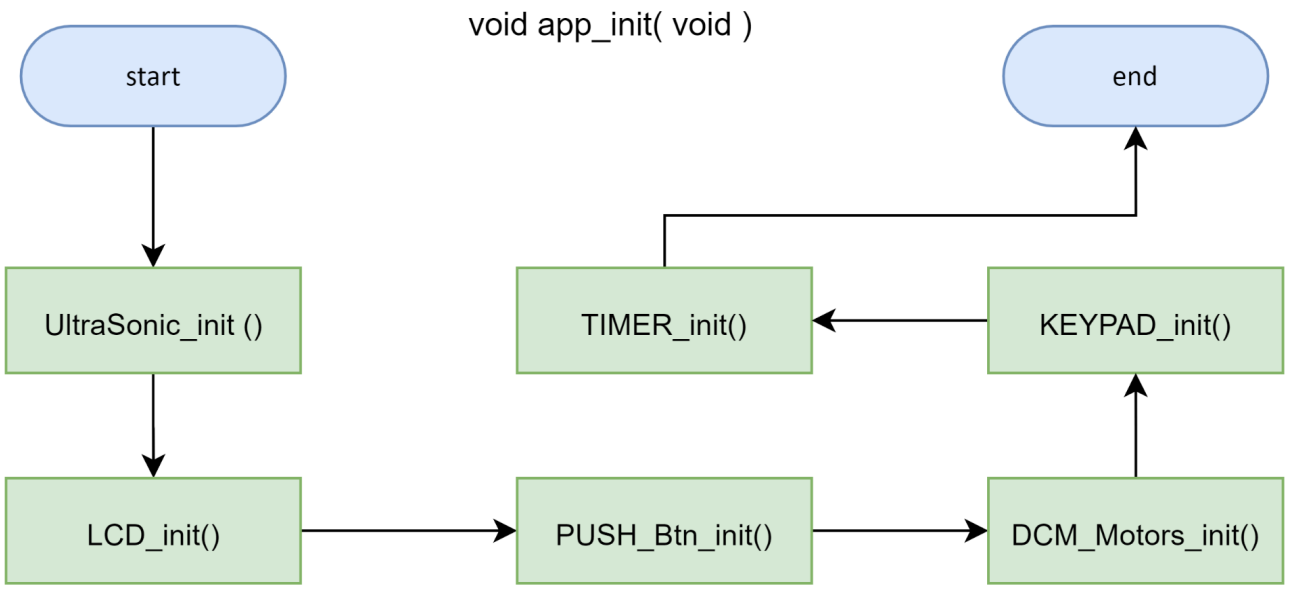


3.3.1.7.Car_stop

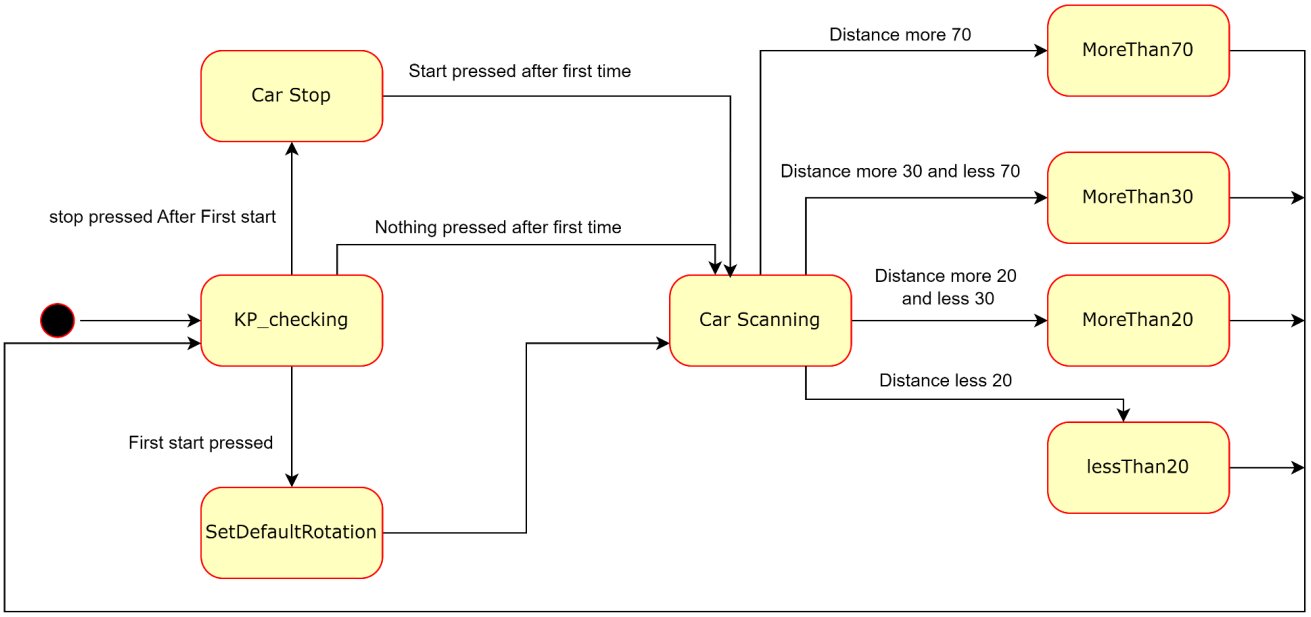


3.3.2. App

3.3.2.1.app_init



3.3.2.2.app_main



4. Precompiling and linking configuration

4.1.DIO

4.1.1.linking Configuration

```
#ifndef DIO_CONFIG_H
#define DIO_CONFIG_H


// Pin modes
#define DIOMODE_INPUT      0
#define DIOMODE_OUTPUT     1

// Input/output settings
#define DIOOUTPUT_LOW       0
#define DIOOUTPUT_HIGH      1

#define DIOINPUT_FLOATING   0
#define DIOINPUT_PULLUP     1

// Pull-up resistor configurations
#define DIOPULLUP_DISABLED 0
#define DIOPULLUP_ENABLED   1


#endif /* DIO_CONFIG_H */
```

4.1.2.Precompiling Configuration

```c
typedef enum{
    PA=0,
    PB,
    PC,
    PD
}DIO_Port_type;

typedef enum{
    OUTPUT,
    INFREE,
    INPULL
}DIO_PinStatus_type;

typedef enum{
    LOW=0,
    HIGH,
}DIO_PinVoltage_type;


typedef enum{
    PINA0=0,
    PINA1=1,
    PINA2,
    PINA3,
    PINA4,
    PINA5,
    PINA6,
    PINA7,
    PINB0,
    PINB1,
    PINB2,
    PINB3,
    PINB4,
    PINB5,
    PINB6,
    PINB7,
    PINC0,
    PINC1,
    PINC2,
    PINC3,
    PINC4,
    PINC5,
    PINC6,
    PINC7,
    PIND0,
    PIND1,
    PIND2,
    PIND3,
    PIND4,
    PIND5,
    PIND6,
    PIND7,
    TOTAL_PINS
}DIO_Pin_type;
```

55 %

4.2.LCD

4.2.1.Precompiling Configuration

```
#define LCD_CFG_H_



#define M_5X7_2lines          0x38
#define Display_ON_No_C       0x0c
#define Display_ON_C          0x0e
#define Display_ON_C_Blanking 0x0f
#define CLR_INS               0x01
#define increment_No_shift    0X06

#define _4_BIT   0
#define _8_BIT   1

/* _4_BIT OR _8_BIT*/
#define  LCD_MODE  _4_BIT



#define  RS   1
#define  RW   2
#define  EN   3
#define  D7   7
#define  D6   6
#define  D5   5
#define  D4   4

#define  P7   7
#define  P6   6
#define  P5   5
#define  P4   4
#define  P3   3
#define  P2   2
#define  P1   1
#define  P0   0
#define LCD_PORT  DIO_PORTA
```

4.3.Timer

4.3.1 linking Configuration

```c
typedef enum {
    TMR_ENABLED,
    TMR_DISABLED
}TimerEnable_t;

typedef enum {
    TMR_ISR_ENABLED,
    TMR_ISR_DISABLED
}TimerISREnable_t;

typedef enum {
    TMR_MODULE_CLK,
    TMR_RISING_EDGE,
    TMR_FALLING_EDGE,
}TimerClockMode_t;

typedef enum {
    TMR_NORMAL_PORT_OPERATION_OC_PIN_DISCONNECTED,
    TMR_TOGGLE_OC_PIN_ON_COMPARE_MATCH,
    TMR_CLEAR_OC_PIN_ON_COMPARE_MATCH,
    TMR_SET_OC_PIN_ON_COMPARE_MATCH
}TimerCompMatchOutputMode_t;

typedef enum
{
    TIMER0,                         /**< Timer 0*/
    TIMER1,                         /**< Timer 1*/
    TIMER2,                         /**< Timer 2*/
    MAX_TIMERS                      /**< Timers on the microcontroller*/
}TimerChannel_t;

typedef enum
{
    TMR_NO_CLOCK_SOURCE_TIMER_COUNTER_ATOPPED,
    TMR_NO_PRESCALING,
    TMR_CLK_8_FROM_PRESCALER,
    TMR_CLK_64_FROM_PRESCALER,
    TMR_CLK_256_FROM_PRESCALER,
    TMR_CLK_1024_FROM_PRESCALER,
    TMR_EXT_CLOCK_SOURCE_ON_T1_PIN_CLK_ON_FALLING_EDGE,
    TMR_EXT_CLOCK_SOURCE_ON_T1_PIN_CLK_ON_RISING_EDGE
}TimerPrescaler_t;
```

```c
    (uint8_t*)TIMER_U8_ICCR1B_REG
};*/
/*****************************************************************/
/**
* Defines a table of pointers to the peripheral TCNT register on the
* microcontroller.
*/
/*static uint8_t volatile * const TmrTCNT_reg[NUMBER_OF_PORTS] =
{
    (uint8_t*)TMR_U8_TCNT0_REG,
    (uint8_t*)TMR_U8_TCNT1_REG,
    (uint8_t*)TMR_U8_TCNT2_REG
};*/
/*****************************************************************/

// uint8_t volatile *const tmrReg[NUM_TIMERS] =
//        {
//                (uint8_t*)TIMER_U8_TCCR_REG,    /*Timer/Counter Control Register*/
//                (uint8_t*)TIMER_U8_TCNT_REG,    /*Timer/Counter Register*/
//                (uint8_t*)TIMER_U8_OCR_REG,     /* Output Compare Register*/
//                (uint8_t*)TIMER_U8_TIMSK_REG,  /*Timer/Counter Interrupt Mask Register*/
//                (uint8_t*)TIMER_U8_TIFR_REG,   /*Timer/Counter Interrupt Flag Register*/
//               // (uint8_t*)&SFIOR,  /* PSR10: Prescaler Reset Timer/Counter1 and Timer/Counter0*/
//        };

const TimerConfig_t TimerConfig[]=
{
    {  TIMER0, TMR_DISABLED,  TMR_OVERFLOW_MODE, TMR_NORMAL_PORT_OPERATION_OC_PIN_DISCONNECTED,  TMR_INTERNAL,  TMR_MODULE_CLK, TMR_CLK_1024_FROM_PRESCALER, TMR_ISR_ENABLED },
    {  TIMER1, TMR_DISABLED,  TMR_OVERFLOW_MODE, TMR_NORMAL_PORT_OPERATION_OC_PIN_DISCONNECTED,  TMR_INTERNAL,  TMR_MODULE_CLK, TMR_CLK_1024_FROM_PRESCALER, TMR_ISR_ENABLED },
    {  TIMER2, TMR_DISABLED,  TMR_OVERFLOW_MODE, TMR_NORMAL_PORT_OPERATION_OC_PIN_DISCONNECTED,  TMR_INTERNAL,  TMR_MODULE_CLK, TMR_CLK_1024_FROM_PRESCALER, TMR_ISR_ENABLED }
};
```

4.3.2 Precompiling Configuration

```c
/* ****************************************************************/
/* TIMER Includes */

/* STD LIB */
#include <math.h>

/* Common */
#include "../../COMMON/BIT_Math.h"
#include "../../COMMON/STD_Types.h"
#include "../../COMMON/vect_table.h"
/* ****************************************************************/
/* TIMER Macros */

#define F_CPU                                1000000UL
/* The 3 Timers counted from 0 to 2 */
#define TIMER_U8_TIMER0                             0
#define TIMER_U8_TIMER1                          1
#define TIMER_U8_TIMER2                          2

/* The 3 Timers Channels counted from 0 to 2 */
#define TIMER_U8_NO_CHANNEL                      0
#define TIMER_U8_TIMER_1_CHANNEL_A                  1
#define TIMER_U8_TIMER_1_CHANNEL_B                  2

/* The 3 Timers Interrupts counted from 0 to 2 */
#define TIMER_U8_CAPT_INTERRUPT                  0
#define TIMER_U8_COMP_INTERRUPT                  1
#define TIMER_U8_OVF_INTERRUPT                  2

/* The 3 Timers Compare Match Output Modes counted from 0 to 2 */
#define TIMER_U8_TOG_OCR_PIN                     0
#define TIMER_U8_CLR_OCR_PIN                     1
#define TIMER_U8_SET_OCR_PIN                     2

/* ****************************************************************/

typedef enum {
    INTERRUPT,
    POLLING
    }EN_TIMER_mode_T;
```

4.4. ICU

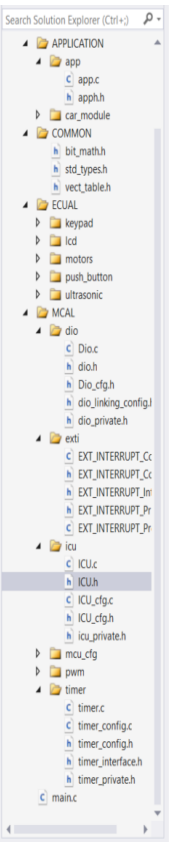
4.4.1 linking Configuration

```c
#ifndef ICU_CFG_H_
#define ICU_CFG_H_
/* ICU_cfg typedefs */

/* The 3 External Interrupts counted from 0 to 2 */
typedef enum {
    EXI_U8_INT0,
    EXI_U8_INT1,
    EXI_U8_INT2
}EN_ICU_usedExti_t;

/* Interrupts Sense Control */
typedef enum {
    EXI_U8_SENSE_LOW_LEVEL,
    EXI_U8_SENSE_LOGICAL_CHANGE,
    EXI_U8_SENSE_FALLING_EDGE,
    EXI_U8_SENSE_RISING_EDGE
}EN_ICU_senseControl_t;

/* Interrupts Sense Control */
typedef enum {
    ISR_DISABLED,
    ISR_ENABLED
}EN_ICU_timer1ISR_t;

typedef struct {
    EN_ICU_usedExti_t ICU_exti;
    EN_ICU_senseControl_t ICU_firstSenseControl;
    EN_ICU_senseControl_t ICU_secondSenseControl;
    EN_ICU_timer1ISR_t timer1_ISR;
}ST_ICU_g_Config_t;
```

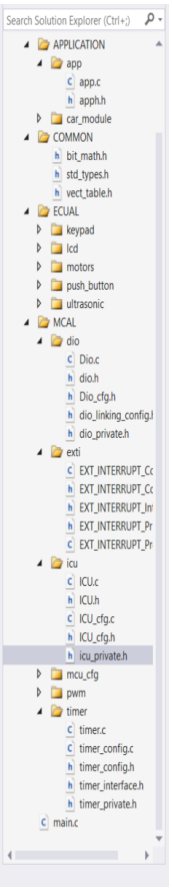
```c
/*
 * ICU_cfg.c
 *
 * Created: 5/16/2023 8:24:24 PM
 *        Author: Mahmoud Mowafey - https://github.com/Mahmoud-Mowafy
 *  Description: This file contains all Direct Current Motor (DCM) functions' implementation.
 *      MCU Datasheet: AVR ATmega32
 *                https://ww1.microchip.com/downloads/en/DeviceDoc/Atmega32A-DataSheet-Complete-DS40002072A.pdf
 */

#include "ICU_cfg.h"
ST_ICU_g_Config_t ST_g_softwareICU[1] =
{
   { EXI_U8_INT2 ,  EXI_U8_SENSE_RISING_EDGE , EXI_U8_SENSE_FALLING_EDGE, ISR_ENABLED}
};
```

4.4.2 Precompiling Configuration

```c
/*
 * ICU.h
 *
 * Created: 5/16/2023 8:25:05 PM
 *         Author: Mahmoud Mowafey - https://github.com/Mahmoud-Mowafy
 *  Description: This file contains all Direct Current Motor (DCM) functions' implementation.
 *       MCU Datasheet: AVR ATmega32
 *                 https://ww1.microchip.com/downloads/en/DeviceDoc/Atmega32A-DataSheet-Complete-DS40002072A.pdf
 */


#ifndef ICU_H_
#define ICU_H_

#include "../../COMMON/BIT_Math.h"
#include "../../COMMON/STD_Types.h"
#include "../../COMMON/vect_table.h"
#include "ICU_cfg.h"
#include "icu_private.h"
#include "../timer/timer_interface.h"
#include "../timer/timer_private.h"

extern ST_ICU_g_Config_t ST_g_softwareICU[1];

typedef enum {
    RISING,
    FALLING
    }EN_icuEdgeFlag;


void ICU_RisingEdgeCapture(void);
void ICU_getValue(Uint32_t *var);

void EXI_enablePIE( Uchar8_t u8_a_interruptId, Uchar8_t u8_a_senseControl );
/*void ICU_FallingEdgeCapture(void);*/
EN_TIMER_ERROR_T TIMER_tmr1NormalModeInit(EN_TIMER_INTERRPUT_T en_a_interrputEnable);
void TIMER_tmr1ClearCompMatInit(void);
EN_TIMER_ERROR_T TIMER_tmr1Start(Uint16_t u16_a_prescaler);
void TIMER_tmr1Stop(void);

#endif /* ICU_H_ */
```



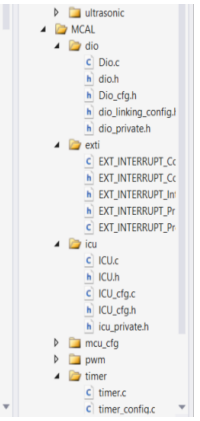

```c
/*
 * icu_private.h
 *
 * Created: 5/17/2023 5:56:15 AM
 *         Author: Mahmoud Mowafey - https://github.com/Mahmoud-Mowafy
 *  Description: This file contains all Direct Current Motor (DCM) functions' implementation.
 *       MCU Datasheet: AVR ATmega32
 *                 https://ww1.microchip.com/downloads/en/DeviceDoc/Atmega32A-DataSheet-Complete-DS40002072A.pdf
 */


#ifndef ICU_PRIVATE_H_
#define ICU_PRIVATE_H_

/*
 * 16-bit Timer/Counter1
 */

#define TMR_U8_TCCR1A_REG      *( ( volatile  u8 * ) 0x4F )
#define TMR_U8_TCCR1B_REG      *( ( volatile  u8 * ) 0x4E )
#define TMR_U8_TCNT1H_REG      *( ( volatile  u8 * ) 0x4D )
#define TMR_U8_TCNT1L_REG      *( ( volatile  u8 * ) 0x4C )
 /* DataType is u16 * in order to get both registers ( i.e. TCNT1L and TCNT1H respectively ) locations in memory */
#define TMR_U16_TCNT1_REG      *( ( volatile  u16 * ) 0x4C )
#define TMR_U8_OCR1AH_REG      *( ( volatile  u8 * ) 0x4B )
#define TMR_U8_OCR1AL_REG      *( ( volatile  u8 * ) 0x4A )
/* DataType is u16 * in order to get both registers ( i.e. OCR1AL and OCR1AH respectively ) locations in memory */
#define TMR_U16_OCR1A_REG      *( ( volatile  u16 * ) 0x4A )
#define TMR_U8_OCR1BH_REG      *( ( volatile  u8 * ) 0x49 )
#define TMR_U8_OCR1BL_REG      *( ( volatile  u8 * ) 0x48 )
/* DataType is u16 * in order to get both registers ( i.e. OCR1BL and OCR1BH respectively ) locations in memory */
#define TMR_U16_OCR1B_REG      *( ( volatile  u16 * ) 0x48 )
#define TMR_U8_ICR1H_REG       *( ( volatile  u8 * ) 0x47 )
#define TMR_U8_ICR1L_REG       *( ( volatile  u8 * ) 0x46 )
/* DataType is u16 * in order to get both registers ( i.e. ICR1L and ICR1H respectively ) locations in memory */
#define TMR_U16_ICR1_REG       *( ( volatile  u16 * ) 0x46 )

/*
 * 16-bit Timer/Counter1 Registers' Description
 */

 /* Timer/Counter1 Control Register A - TCCR1A */
 /* Bit 7:6 - COM1A1:0: Compare Output Mode for Channel A */
#define TMR_U8_COM1A1_BIT      7
#define TMR_U8_COM1A0_BIT      6
/* Bit 5:4 - COM1B1:0: Compare Output Mode for Channel B */
#define TMR_U8_COM1B1_BIT      5
#define TMR_U8_COM1B0_BIT      4
/* Bit 3 - FOC1A: Force Output Compare for Channel A */
#define TMR_U8_FOC1A_BIT       3
/* Bit 2 - FOC1B: Force Output Compare for Channel B */
#define TMR_U8_FOC1B_BIT       2
/* Bit 1:0 - WGM11:0: Waveform Generation Mode */
#define TMR_U8_WGM11_BIT       1
#define TMR_U8_WGM10_BIT       0
/* End of TCCR1A Register */

/* Timer/Counter1 Control Register B - TCCR1B */
/* Bit 7 - ICNC1: Input Capture Noise Canceler */
#define TMR_U8_ICNC1_BIT       7
/* Bit 6 - ICES1: Input Capture Edge Select */
#define TMR_U8_ICES1_BIT       6
```



4.5. DCM

4.5.1 Linking Configuration

```c
 /* HAL */
 #include "../../MCAL/dio/dio_linking_config.h"
#include "DCM_private.h"
#include "dcm_cfg.h"

/*************************************************************************************************************************************
/* Declaration and Initialization */


ST_DCM_g_Config_t ST_g_carMotors[2]=
{
    {    MOT0_EN_PIN_NUMBER_0 ,  MOT0_EN_PIN_NUMBER_1, MOT0_EN_PORT_NUMBER},
    {    MOT1_EN_PIN_NUMBER_0 ,  MOT1_EN_PIN_NUMBER_1, MOT1_EN_PORT_NUMBER}
};
```



4.5.1 Precompiling Configuration

```c

extern ST_DCM_g_Config_t ST_g_carMotors[2];
/*************************************************************************************************************************************
/* DCM Functions' Prototypes */

EN_DCM_ERROR_T DCM_rotateDCM(EN_DCM_MOTORSIDE DCM_l_motorNumber, u16 DCM_a_rotateSpeed);

EN_DCM_ERROR_T DCM_changeDCMDirection(ST_DCM_g_Config_t* DCM_a_ptrToConfig, EN_DCM_MOTORSIDE DCM_a_motorNum);

EN_DCM_ERROR_T DCM_u8SetDutyCycleOfPWM(u8 DCM_a_dutyCycleValue);

void DCM_vdStopDCM(void);

EN_DCM_ERROR_T DCM_motorInit(ST_DCM_g_Config_t* DCM_a_ptrToConfig);

void DCM_updateStopFlag(void);
void DCM_MoveForward(u8 u8_a_speed);
void DCM_MoveBackward(u8 u8_a_speed);

//u8 DCM_u8GetDutyCycleOfPWM(u8* Cpy_pu8ReturnedDutyCycleValue);
/*************************************************************************************************************************************

/*************************************************************************************************************************************

```

4.6. PWM

4.6.1 Linking Configuration

```
typedef struct ST_PWM_PINS_CONFIGS{


    EN_DIO_Pin_type en_pwm_pin ;


}ST_PWM_PINS_CONFIGS;

const ST_PWM_PINS_CONFIGS st_pwm_configs[PWM_PINS_NUMBER] =
{
    {
        PIND4
    },
    {
        PIND5
    }
};


};
```

4.6.2 Precompiling Configuration

```
#define TIMER_STOPPED                              0
#define TIMER_NO_PRESCALER                         1
#define TIMER_8_PRESCALER                          2
#define TIMER_64_PRESCALER                         3
#define TIMER_256_PRESCALER                        4
#define TIMER_1024_PRESCALER                       5
#define TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE   6
#define TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE    7
/*Timer Prescaler Options:
 * 0- TIMER_STOPPED
 * 1- TIMER_NO_PRESCALER
 * 2- TIMER_8_PRESCALER
 * 3- TIMER_64_PRESCALER
 * 4- TIMER_256_PRESCALER
 * 5- TIMER_1024_PRESCALER
 * 6- TIMER_EXTERNAL_CLOCK_SOURCE_FALLING_EDGE
 * 7- TIMER_EXTERNAL_CLOCK_SOURCE_RISING_EDGE
 */
#define TIMER_SET_PRESCALER    TIMER_1024_PRESCALER

#define  REG_SIZE                                  256

/*configuration of pwm pins*/
#define PWM_PINS_NUMBER     2
```

4.7. Keypad

4.7.1 Linking Configuration

```
/* enum to define all keys we can change it according to our keypad size (we need two keys only) */
typedef enum EN_KEYPAD_KEYS{
    KEY_NOTHING = 0 ,
    KEY_1 ,
    KEY_2 ,
}EN_KEYPAD_KEYS;

/* this struct contain all keypad configs */
typedef struct ST_KEYPAD_CONFIG{

    /* array that holds all cols pins */
    ST_DIO_ConfigType u8_arr_cols[NUMBER_OF_COLS];

    /* array that holds all rows pins */
    ST_DIO_ConfigType u8_arr_rows[NUMBER_OF_ROWS];

    /* 2D array that holds all keys values according to it's position [row][col] */
    EN_KEYPAD_KEYS u8_arr_keys[NUMBER_OF_ROWS][NUMBER_OF_COLS];
}ST_KEYPAD_CONFIG;
```

```
/* variable of type keypad configs to set our configuration */
const ST_KEYPAD_CONFIG st_keypad_conf =
{
    .u8_arr_cols =
    {
        {
            .dio_port = KEPAD_PORT,
            .dio_pin  = C1,
            .dio_mode = DIO_MODE_INPUT,
            .dio_initial_value  = DIOINPUT_PULLUP,
        },
        {
            .dio_port = KEPAD_PORT,
            .dio_pin  = C2,
            .dio_mode = DIO_MODE_INPUT,
            .dio_initial_value  = DIOINPUT_PULLUP,
        }},

    .u8_arr_rows =
    {{
            .dio_port = KEPAD_PORT,
            .dio_pin  = R1,
            .dio_mode = DIO_MODE_OUTPUT,
            .dio_initial_value  = DIO_HIGH,
            .dio_pullup_resistor = DIO_PULLUP_DISABLED

        }},
    .u8_arr_keys ={{KEY_1 , KEY_2}}
};
```

4.7.2 Precompiling Configuration

```
#define KEPAD_PORT  DIO_PORTC

/* according to number of rows you need to define Rx pins here we have just one row so we define R1 only*/
#define NUMBER_OF_ROWS 1
#define R1    2
/* according to number of cols you need to define Cx pins here we have just two cols so we define C1 and C2*/
#define NUMBER_OF_COLS 2

#define C1 5
#define C2 6
```

4.8 External Interrupt

4.8.1 Precompiling Configuration

```
10
11
12  #define EXT_INTERRUPT_PINS          1
13
14  typedef enum
15  {
16      MCUCR_REG_ISC00_BITS = 0,
17      MCUCR_REG_ISC01_BITS,
18      MCUCR_REG_ISC10_BITS,
19      MCUCR_REG_ISC11_BITS
20
21  }EN_MCUCR_REG_BITS;
22
23  typedef enum
24  {
25      MCUCSR_REG_ISC2_BITS = 6,
26
27  }EN_MCUCSR_REG_BITS;
28
29  typedef enum
30  {
31      GICR_REG_INT2_BITS = 5,
32      GICR_REG_INT0_BITS,
33      GICR_REG_INT1_BITS
34
35  }EN_GICR_REG_BITS;
36
```

```
34
35  }EN_GICR_REG_BITS;
36
37  typedef enum
38  {
39      GIFR_REG_INTF2_BITS = 5,
40      GIFR_REG_INTF0_BITS,
41      GIFR_REG_INTF1_BITS
42
43  }EN_GIFR_REG_BITS;
44
45  typedef enum
46  {
47      LOW_LEVEL_SENSE_CONTROL = 0,
48      ANY_LOGICAL_SENSE_CONTROL,
49      FALLING_EDGE_SENSE_CONTROL,
50      RISING_EDGE_SENSE_CONTROL
51
52  }EN_EXT_INTERRUPT_Sense_Control;
53
54  typedef enum
55  {
56      EXT0_INTERRUPTS = 0,
57      EXT1_INTERRUPTS,
58      EXT2_INTERRUPTS
59  }EN_EXT_INTERRUPTS;
```

4.8.2 Linking Configuration

```
60
61  typedef struct
62  {
63      void(*INTERRUPT_EXTERNAL_HANDLER)(void);
64      EN_EXT_INTERRUPTS EXTERNAL_INTERRUPRT_Number;
65      EN_EXT_INTERRUPT_Sense_Control EXTERNAL_INTERRUPRT_Sense_Control;
66
67  }ST_EXT_INTERRUPTS_CFG;
68
```

4.9 Push Button

4.9.1 Precompiling Configuration

```
13
14  #define  NUMBER_OF_PUSH_BUTTONS          1
15
16  typedef enum
17  {
18      PUSH_BTN_STATE_PRESSED = 0,
19      PUSH_BTN_STATE_RELEASED
20  }EN_PUSH_BTN_state_t;
21
22  typedef enum
23  {
24      PUSH_BTN_PULL_UP = 0,
25      PUSH_BTN_PULL_DOWN
26  }EN_PUSH_BTN_active_t;
27
```

4.9.2 Linking Configuration

```
29
30  typedef struct
31  {
32      ST_DIO_ConfigType PUSH_BTN_pin;
33      EN_PUSH_BTN_state_t PUSH_BTN_state;
34      EN_PUSH_BTN_active_t PUSH_BTN_connection;
35  }ST_PUSH_BTN_t;
```

```
9
10  const ST_PUSH_BTN_t A_pbConfig[NUMBER_OF_PUSH_BUTTONS] =
11  {
12      {
13          .PUSH_BTN_pin.dio_mode = DIO_MODE_INPUT,
14          .PUSH_BTN_pin.dio_pin = DIO_PIN4,
15          .PUSH_BTN_pin.dio_port = DIO_PORTC,
16          .PUSH_BTN_pin.dio_initial_value = DIO_HIGH,
17          .PUSH_BTN_pin.dio_pullup_resistor = DIO_PULLUP_ENABLED,
18          .PUSH_BTN_connection = PUSH_BTN_PULL_UP,
19          .PUSH_BTN_state = PUSH_BTN_STATE_RELEASED
20      }
21  };
```

4.10 Ultrasonic

4.10.1 Linking Configuration

```
18
19  typedef struct
20  {
21      ST_DIO_ConfigType triggerpin;
22      ST_DIO_ConfigType echopin;
23  }ST_ultrasonicPins;
24
```